VŠB – Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

DIPLOMOVÁ PRÁCE

2020 Bc. Jiří Frank

VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Mobilní aplikace pro zobrazení signálů detektoru poruch vodičů vysokého napětí

Mobile Application to Display Signals of Medium Voltage Overhead Lines Fault Detector

2020 Bc. Jiří Frank

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Jiří Frank**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Mobilní aplikace pro zobrazení signálů detektoru poruch vodičů vysokého napětí
Mobile Application to Display Signals of Medium Voltage Overhead Lines Fault Detector

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je navrhnout a naimplementovat mobilní aplikaci určenou pro operační systém Android. Aplikace bude sloužit pro uživatelsky přívětivou práci se signály naměřenými detektorem poruch na vedení vysokého napětí. Součástí aplikace bude vlastní implementace komponenty pro zobrazení signálů.

Práce bude splňovat následující body:
1. Stručný popis platformy OS Android.
2. Popis metod webové služby poskytující naměřené signály.
3. Návrh a implementace komponenty pro zobrazení signálů.
4. Návrh a implementace aplikace pro práci se signály včetně zobrazení a úpravy údajů vztažených k signálům.
5. Vytvoření instalačního balíčku a uživatelské dokumentace.

Seznam doporučené odborné literatury:

[1] ULLMAN, Jeffrey D.; WIDOM, J. Database Systems: The Complete Book. 2000.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Petr Lukáš**

Datum zadání: 01.09.2019
Datum odevzdání: 30.04.2020

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky 1D16 OSTRAVA

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 5. května 2020

Frank

.....

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 5. května 2020

Frank

.....

Rád bych na tomto místě poděkoval vedoucímu diplomové práce, jímž je vážený pan
Ing. Petr Lukáš, Ph.D., který se zapříčinil vznesenými poznámkami a návrhy o zvýšení kvality
této diplomové práce.

Abstrakt

Tato diplomová práce vznikla ve spolupráci s centrem ENET za účelem vývoje detektoru poruch na izolovaném vedení vysokého napětí.

Pro tyto účely již existuje aplikace pro platformu Windows, která přináší veškeré nevýhody a omezení standardních desktopových aplikací. Technici používají současnou aplikaci pro zobrazení detailů o měřeních a převážně k vizualizaci naměřeného signálu.

Cílem diplomové práce je navrhnout a implementovat mobilní aplikaci určenou pro operační systém Android. Tato aplikace má za cíl umožnit uživateli pracovat s naměřenými signály a procházet je.

Klíčová slova: Android, Java, Kotlin, měření průběhu signálů, zobrazení průběhu signálů

Abstract

This diploma thesis was created in cooperation with the ENET center in order to develop a fault detector on an isolated high voltage line.

Applications for this purpose already exist for the Windows platform. This application has the disadvantages and limitations of standard desktop applications. The technical specialists are using the application to display the details of the measurement and mainly visualizes the measured signal.

The aim of the diploma thesis is to design and implement a mobile application designed for the Android operating system. This application aims to allow the user to work with measured signals and scroll through them.

Keywords: Android, Java, Kotlin, measuring the course of signals, display of signals

Obsah

Seznam použitých zkratek a symbolů **10**

Seznam obrázků **11**

Seznam tabulek **12**

Seznam výpisů zdrojového kódu **13**

1 Úvod **14**
1.1 Současný stav 14

2 Platforma Android **17**
2.1 Android a jeho historie 17
2.2 Systémová architektura 18
2.3 Android - verze Application Programming Interface 19
2.4 Aktivita 20
2.5 Návrhové vzory pro UI 21
2.6 Android Studio 23
2.7 Kotlin a Java 24

3 Návrh aplikace **29**
3.1 Datová vrstva – popis webové služby 29
3.2 Datová vrstva - kSOAP2, DataHandler 33
3.3 Android komponenta GraphComponent 37
3.4 Navržené aktivity 50

4 Testování aplikace **53**
4.1 Využití hardware 53
4.2 Výkonnostní srovnání 55

5 Uživatelská dokumentace **56**
5.1 Výběr měřeni 56
5.2 Otevřené měření 58
5.3 Ovládání grafu 59
5.4 Uživatelská nabídka 59

6 Výroba instalačního balíčku **61**

7 Závěr **62**

Seznam použitých zkratek a symbolů

HW	–	Hardware
PC	–	Personal Computer
OS	–	Operating System
JVM	–	Java Virtual Machine
DVM	–	Dalvik Virtual Machine
API	–	Application Programming Interface
SDK	–	Software Development Kit
WCF	–	Windows Communication Foundation
MVC	–	Model View Controller
MVP	–	Model View Presenter
MVVM	–	Model View View-Model
ARM	–	Advanced RISC Machine
AS	–	Android Studio
IP	–	Internet Protocol
FPS	–	Frames Per Second
APK	–	Android Application Package
GPS	–	Global Positioning System
WPF	–	Windows Presentation Foundation
XAML	–	Extensible Application Markup Language

Seznam obrázků

1 Výběr měření u desktop verze 16
2 Mobile OS Market Share [3] 17
3 Android Activity Lifecycle [7] 20
4 MVC diagram 22
5 Struktura projektu 24
6 Datová vrstva 34
7 Základní zobrazení komponenty 37
8 Proces přiblížení 39
9 Základní zobrazení komponenty 40
10 Kreslení ve velkém měřítku 50
11 Kreslení v malém měřítku 50
12 Měření využití hardware 54
13 Výchozí stav aplikace 56
14 Výběr stanice 57
15 Výběr stanice 57
16 Otevřené měření 58
17 Otevřené měření 58
18 Otevřená uživatelská nabídka 59
19 Otevřené konfigurační okno 59
20 Otevřené okno s detaily 60
21 Otevřené okno s anotacemi 60

Seznam tabulek

1 Metody webové služby 29
2 Parametry objektu třídy `StationType` 30
3 Vstupní parametry `GetMetadataFromTo` 30
4 Parametry objektu třídy `PdmetadataType` 31
5 Vstupní parametry `GetFaultAnnotation` 31
6 Parametry objektu třídy `PdpatternType` 31
7 Vstupní parametry `GetFaultAnnotation` 32
8 Vstupní parametry `SettFaultAnnotation` 32
9 Metody třídy `DataHandler` 36
10 Konstruktor `GraphComponent` 42
11 Srovnání aplikací 55

Seznam výpisů zdrojového kódu

1 Ukázka třídy v jazyce Java 26
2 Ukázka třídy v jazyce Kotlin 28
3 Ukázka použití komponenty GraphComponent 41
4 Výpočet počátečního indexu při posunu 44
5 Výpočet pozice okna pro vykreslování 45
6 Chování metody drawOptimizedDataset 47
7 Chování metody drawBigScaleForSpecificPixel 48
8 Výpočet mapy bodů pro snímek v malém měřítku 49

1 Úvod

Mobilní zařízení se stala nedílnou součástí našich životů. Je přirozené, že se tento fenomén netýká pouze soukromých osob, nýbrž i různých pracovníků napříč všemi obory. Dostal jsem možnost, pracovat na této diplomové práci s nadějí, že se budu podílet na urychlení tohoto fenoménu tam, kde to má smysl. Konkrétně se jedná o realizaci návrhu a implementace aplikace, pro mobilní zařízení se systémem Android, která umožní uživateli pracovat s naměřenými signály na vedení vysokého napětí.

V zalesněných oblastech se v těžko dostupném terénu používají pro vedení vysokého napětí izolované vodiče. Díky izolaci mohou být jednotlivé vodiče pro fáze umístěny blíže u sebe, takže se snižují nároky na prostor u takového vedení. Vedení je méně náchylné například k poruchám vzniklým spadlými stromy, spadlými větvemi nebo pádem vodiče na zem. Na běžném vedení bez izolace by podobné problémy mohly vést ke zkratu. Při použití izolace zkrat nenastane, což znamená, že vedení zůstává po nehodě většinou funkční, nicméně i tak je nutné poruchu detekovat.

K této detekci se používá analýza signálů naměřeného napětí. Při poruše se v těchto signálech objevují hroty (tzn. prudké výkyvy), které jsou důkazem o takzvaných částečných výbojích. Problém ale je ten, že hroty se objevují i v signálech, které neznamenají poruchu a rozeznat poruchový a bezporuchový signál může být poměrně obtížné [1]. Z tohoto důvodu je mimo jiné nutné naměřené signály vizuálně zkoumat a analyzovat.

Přínosem práce bude vytvoření mobilní aplikace pro vizualizaci naměřených signálů, což dokáže zefektivnit a zjednodušit práci techniků, kteří mají na starosti řešení poruch přímo v terénu. Dalším přínosem bude vznik Android komponenty, která bude mít na starosti vykreslování průběhů signálů. Doposud totiž neexistuje žádná snadno použitelná komponenta, či knihovna třetí strany, která by efektivně dokázala bez ztráty dat zprostředkovat prezentaci měření, která obvykle přesahují několik stovek tisíc naměřených hodnot.

1.1 Současný stav

Než popíšeme současný stav, je potřeba definovat, co je to *měření*. Měření definujeme jako soubor několika průběhů, kde každý průběh představuje měření napětí v čase. Jedno měření obsahuje 3 až 4 průběhy, které odpovídají jednotlivým fázím třífázového vedení. Čtvrtý průběh, je-li k dispozici, představuje kontrolní měření, které v následujícím textu zjednodušeně nazýváme jako 4. fázi[1]. Jeden průběh obvykle představuje jednu periodu. Při fázové frekvenci 50 Hz tedy jde o 20 ms. Pro měření napětí se používá vzorkovací frekvence 40 MHz. Jeden průběh tedy obvykle zahrnuje 800 000 vzorků.

Samotné měření se provádí na měřících stanicích, které jsou reálně rozmístěny po území ČR. Na těchto stanicích dochází k zaznamenání měření jednou za hodinu. Jednou za hodinu

[1] Jedná se o nepřesný pojem, nicméně je takto techniky používán.

se také spouští proces, který zařídí stažení dat a nahrání těchto dat prostřednictvím webové služby do databáze. Současně dochází i k detekci a případnému nahlášení případných poruch. K tomuto účelu slouží klasifikátor, který je stále ve vývoji. Pro aplikaci jsou poté data dostupná prostřednictvím webové služby.

V aplikaci má uživatel možnost procházet signály dle libosti. Všechna potřebná data, jako seznamy dostupných stanic, metadata ke stanicím a měřením, jednotlivé měření, či anotace, jsou k dispozici prostřednictvím webové služby, viz. kapitola 3.1.

Aplikace jako taková funguje dle očekávání správně, čeká se od ní intuitivní práce se signály s okamžitou odezvou, nicméně i tak sebou přináší nevýhody desktopové aplikace ve srovnání s mobilní aplikací. Tyto nevýhody jsou:

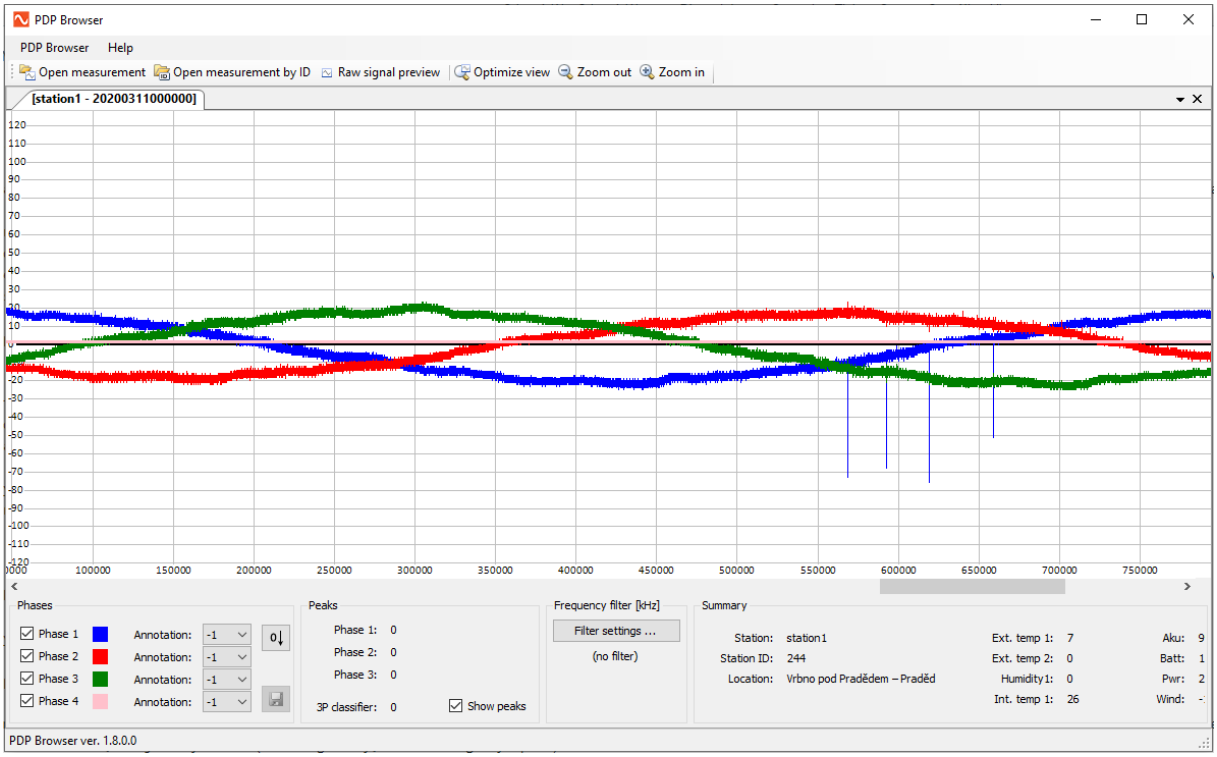
- snížená mobilita,
- závislost na pevném nebo WiFi připojení, v případě laptopu existuje možnost mobilního připojení, nicméně tato možnost je nepraktická, obzvláště v terénu,
- android nativní aplikace nabízí možnosti, které u PC verze nejsou realizovatelné, jako například práci v odpojeném režimu[2],
- závislost na platformě Windows, protože se jedná o Windows Forms [3] aplikaci.

Současná podoba Windows Forms aplikace se nachází na obrázku 1. Pro mobilní aplikaci byla vyčleněna následující akceptační kritéria:

1. Aplikace dokáže konzumovat webové služby.
2. Aplikace umožní uživateli vybrat libovolnou stanici a libovolné měření podle výběru data.
3. Aplikace dokáže uživateli zobrazit anotace fází otevřeného měření.
4. Aplikace umožní uživateli nastavit anotace fází pro otevřené měření.
5. Komponenta pro vykreslení grafu zobrazí všechna měření, tj. jednotlivé fáze bez ztráty dat.
6. Komponenta umožní uživateli snadný a intuitivní pohyb v grafu.
7. Komponenta umožní uživateli dočasně skrýt libovolnou fázi.
8. Komponenta bude při pohybu efektivní a výkonná, nebude docházet k zasekávání popř. jiným potížím s výkonem.
9. Aplikace bude držet grafický styl, zejména barevné schéma, tak, aby nevybočovala z použitého stylu projektu Centra ENET.

[2]Práce v odpojeném režimu je jeden ze zamýšlených konceptů pro vylepšení aplikace, v současné verzi mobilní aplikace není implementován.

[3]https://docs.microsoft.com/cs-cz/dotnet/framework/winforms/



Obrázek 1: Výběr měření u desktop verze

Text diplomové práce bude organizován následovně. Kapitola 2 se bude zabývat popisem platformy Android. Následuje kapitola 3, ve které bude popsána datová vrstva aplikace, Android komponenta a popis navržených aktivity. Další kapitola 4 se věnuje testování výkonu aplikace. Součástí práce je i uživatelská dokumentace v kapitole 5. Vytvoření instalačního balíčku je popsáno v 6. Poslední kapitola 7 je věnována závěru.

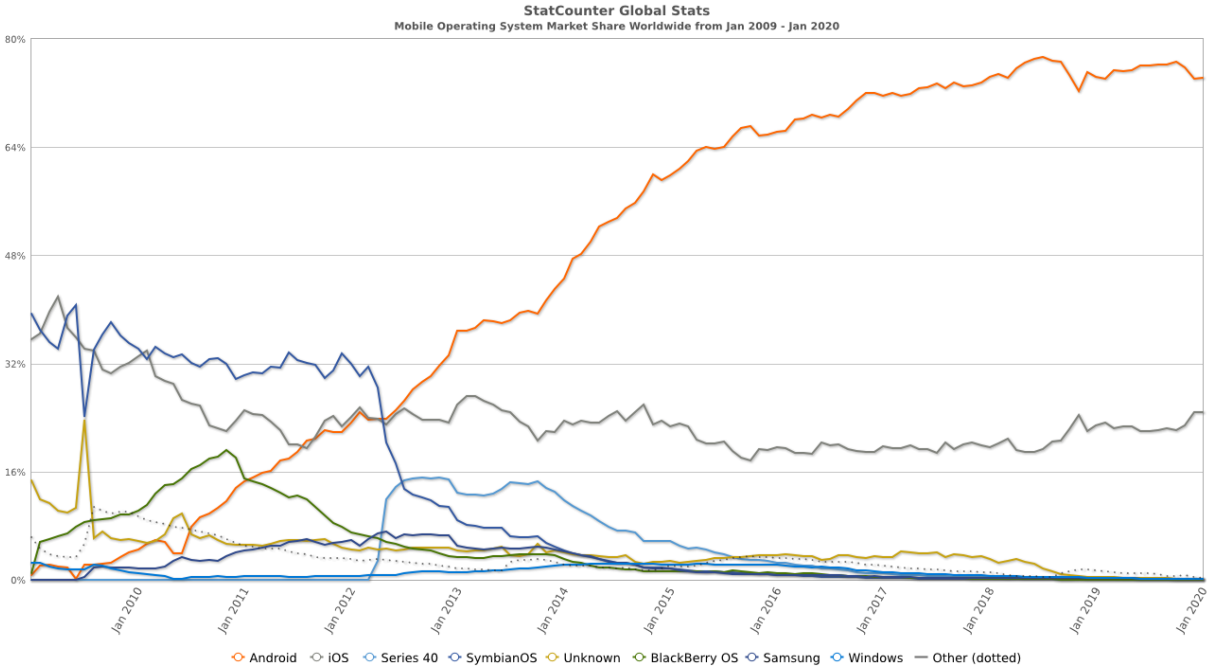
2 Platforma Android

Android, v kontextu OS, je mobilní operační systém. Můžeme jej nalézt na mnoha různých zařízeních. Mezi tato zařízení patří například tablety, chytré telefony, chytré televize, malá přenosná zařízení v podobě chytrých náramků, nebo hodinek. Můžeme říct, že právě Android získal největší zastoupení v podílu trhu co do počtu zmíněných zařízení a jejich dalších adaptací viz. kapitola 2.1.

2.1 Android a jeho historie

Původní Android vyvinula firma Android, Inc, která se začala formovat v roce 2003. Původními zakladateli byli Chris White, Nick Sears, Rich Miner a Andy Rubin. V podstatě se jednalo o malou začínající společnost, která se původně zabývala operačním systémem pro mobilní zařízení, které dokázalo pracovat s údaji o poloze. Po proniknutí na trh a následných potížích s financováním, se společnost Google rozhodla tuto společnost v roce 2005 odkoupit.

Následnou spoluprací vzniklo v roce 2007 uskupení Open Handset Alliance, což vedlo k vytvoření nového otevřeného mobilního standardu. První komerčně uvedený chytrý telefon, postavený na těchto nově vzniklých standardech byl představen v roce 2008. Můžeme tedy konstatovat, že v tomto roce byly položeny první základy éry Androidu [2]. Podle dat, dostupných na plat-



Obrázek 2: Mobile OS Market Share [3]

formě Statcounter [3],lze jednoznačně vypozorovat historický vývoj podílu OS Android na trhu

s chytrými telefony, který soustavně od jeho vzniku stále roste. V lednu 2020 dosahuje úrovně 74,3 %, viz. Obrázek 2.

2.2 Systémová architektura

Architektura operačního systému Android se skládá z několika vrstev [4].

1. Linuxové jádro - tvoří základ OS Android. Poskytuje základní operace a ve spolupráci s HW ovladači obsluhuje HW komponenty zařízení. Linuxové jádro se stará o:
   - správu paměti,
   - správu procesů,
   - HW ovladače,
   - síťovou komunikaci,
   - bezpečnost.
2. HW Abstraktní vrstva - tato vrstva umožňuje využívat konkrétní HW prvky, jako je například mikrofon a to bez nutnosti znát detail HW a způsob jak jej obsluhovat.
3. Běhové prostředí - spouští nativní Android aplikaci v prostředí virtuálního stroje. Nejedná se o Java Virtual Machine, ale o Dalvik Virtual Machine, více v kapitole 2.2.1.
4. Nativní knihovny - doplňující nativní knihovny k běhovému prostředí.
5. Aplikační Framework - Java API, bežně dostupné při programování.
6. Systémové a uživatelské aplikace - standardní aplikace dostupné v rámci konkretního zařízení. Rozdíl mezi systémovou a uživatelskou aplikací je pouze ten, že systémová aplikace je k dispozici již od výrobce a na rozdíl od uživatelských aplikací přetrvává i po obnově zařízení.

2.2.1 Dalvik Virtual Machine

Zmíněný DVM je virtální stroj, upravený pro mobilní zařízení. Srovnání s klasickým JVM vypadá následovně. JVM je virtuální stroj založený na zásobníku. Znamená to, že každé vlákno má přidělený paměťový prostor v podobě zásobníku. Ten je tvořen takzvanými zásobníkovými rámci. Překlad aplikace vypadá takto.

1. Soubory ve formátu .java jsou zkompilovány,
2. výsledkem je Java bytecode v podobě souborů .class,
3. tzv. Java bytecode potom zpracuje JVM.

Oproti tomu DVM počítá s menším výkonem zařízení, proto je založený na registrech[5][6]. Tento přístup je vhodnější pro typickou ARM architekturu. To obnáší několik výhod:

- vyhnutí se rozbalování instrukcí (fetching/reading),
- redukce přístupů do paměti,
- efektivnější zpracovávání instrukcí.

DVM nezpracovává Java bytecode, nýbrž Dalvik bytecode, který je vytvořen z Java bytecode. V celém procesu je tedy navíc překlad do Dalvik bytecode.

1. Soubory ve formátu .java jsou zkompilovány,
2. výsledkem je Java bytecode v podobě .class,
3. Dex kompilátor zpracuje Java bytecode a vyrobí tzv. Dalvik bytecode v podobě .dex souborů,
4. Dalvik bytecode potom zpracuje DVM.

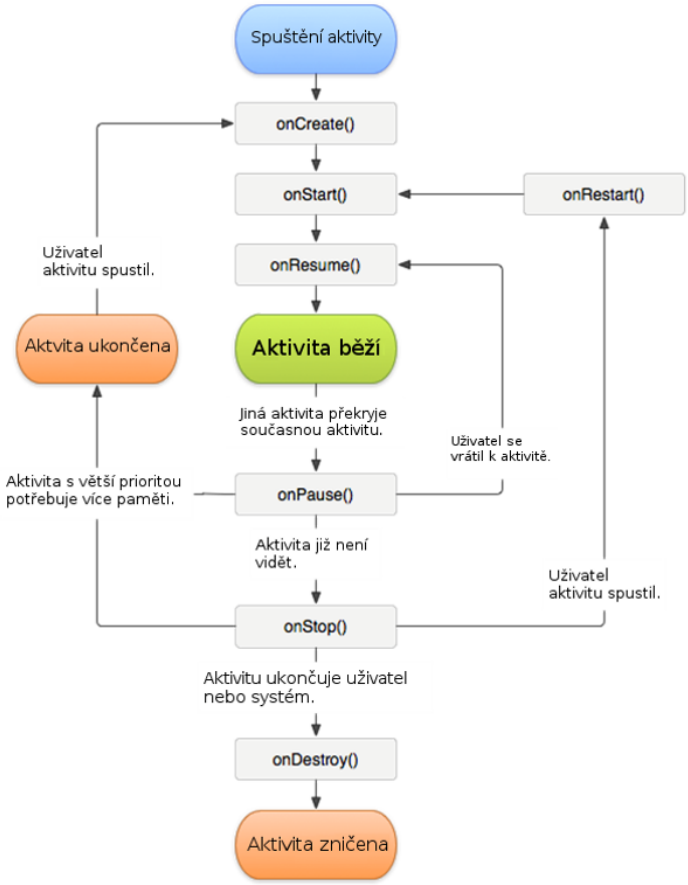
2.3 Android - verze Application Programming Interface

Při vytváření nové Android aplikace je nutno specifikovat, na jakou minimální verzi Android API budeme cílit. Pokud cílený uživatel vlastní zařízení se starší verzí Androidu, respektive API, nepůjde mu aplikace spustit, pokud jeho zařízení má stejnou nebo vyšší verzi, vše by mělo fungovat. Čím novější API zvolíme, tím více novějších, vyladěnějších a mnohdy i jednodušších přístupů či variant řešení různých problémů máme obvykle k dispozici. Zároveň s tím ovšem přicházíme o část uživatelů, kteří nevlastní zařízení s námi zvolenou minimální verzí.

Záleží tedy na správné analýze toho, co od aplikace čekáme a na jak velkou skupinu uživatelů cílíme. Tato aplikace určitě necílí na příliš levné nebo staré telefony a proto lze zvolit jednu z novějších verzí a tou je API 23 tj. verze androidu 6.0+.

2.4 Aktivita

Aktivita je jedním ze základních stavebních kamenů každé aplikace. Jde o obdobu formuláře z aplikací v OS Windows. Ve světě Android ovšem platí trochu jiná pravidla. Aktivita není viditelná formou okna na obrazovce, nýbrž vyplňuje celou obrazovku, podobně jako kdybychom srovnali s klasickým formulářem, ovšem pouze v režimu celé obrazovky. Má na starosti vykreslení a zachytávání různých událostí, které mohou sloužit jako vstup. Aktivita má konkrétní GUI, které je definováno formou XML souborů. Každá aktivita má svůj vlastní životní cyklus, který je popsán na obrázku 3. Obdobný systém definice GUI je použit při vývoji WPF aplikací, kde se používají XAML soubory.



Obrázek 3: Android Activity Lifecycle [7]

Tím, jak měníme jednotlivé aktivity v rámci aplikace, mění se i stav aktivity, dle zobrazeného stavového diagramu na Obrázku 3, z toho vyplývá, že běžící aktivita může být v jednom okamžiku pouze jedna.

Aktivita má několik stavů:

1. **Spuštění aktivity** – aktivita není viditelná, pro přechod do dalšího stavu se musí vykonat metody `onCreate()`, `onStart()` a `onResume()`.

2. **Aktivita běžící** – je viditelná v popředí.

3. **Aktivita ukončena** – jiná aktivita přešla do popředí, stav běžící se mění na ukončena, volá se `onPause()` a `onStop()`, tento stav nastane v případě, že Android potřebuje pro jinou aplikaci více paměti.

4. **Aktivita zničena** – aktivita je ručně uživatelem nebo systémem ukončená, oproti přechodu z běžící do pozastavená se volá navíc `onDestroy()`.

K dispozici máme různé metody, které v rámci vyvíjené aplikace používáme například pro správu chování komponenty. Konkrétním příkladem může být pozastavení vykreslování při `onPause`, v rámci aktivity, na které se nachází komponenta pro vykreslování signálů a `onResume`, kde opět vykreslování obnovujeme.

2.5 Návrhové vzory pro UI

Aktivity android aplikací se obvykle vyvíjí na jednom z MVC návrhových vzorů. Konkrétně jde o:

- MVC – Model View Controller,
- MVP – Model View Presenter,
- MVVM – Model View View-Model.

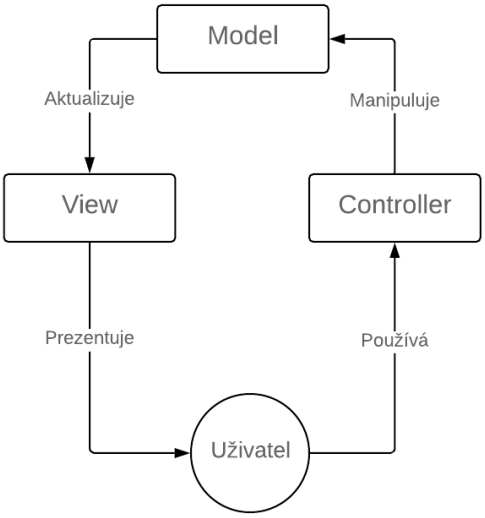
2.5.1 MVC

MVC se skládá ze tří částí:

1. Model – Zahrnuje logiku a přístupy k datům, výsledky změn jsou propagovány do view.
2. View – to co uživatel vidí, podoba je definována v podobě zmíněných XML.
3. Controller – manipuluje s modelem, je zodpovědný za zachycení vstupu a spuštění konkrétních akcí, jedná jako prostředník mezi modelem a view.

MVC je oproti ostatním vzorům vhodnější pro projekty, které nemají rozsáhlé GUI a to hlavně díky jednodušší implementaci. Obecně platí, že pokud má aplikace pouze jednotky komponent, je určitě vhodné použít MVC kvůli menší režii. Do této kategorie spadá i tato mobilní aplikace pro prohlížení signálů.

MVC má i své nevýhody. Jedna z nich je relativně vysoká závislost na API OS Android, dále není modulární a už vůbec není snadné provádět jednotkové testy. Všechny nevýhody jsou zapříčiněny částí Controller, která si kvůli své povaze prostředníka musí vždy svým způsobem pamatovat kontext aplikace jako takové, tedy pamatovat si určitý stav aplikace, a zároveň musí reagovat na uživatelské vstupy. Diagram 4 popisuje architekturu tohoto vzoru.



Obrázek 4: MVC diagram

Důvod zvolení MVC lze snadněji pochopit, když MVC srovnáme s MVVM. MVVM je dle většiny Android vývojářů v současnou chvíli doporučován, jako nejlepší volba, zejména pokud je aplikace rozsáhlejší a zároveň obsahuje rozsáhlé GUI, které s uživatelem reaguje ve větší míře.

MVVM se skládá z těchto částí:

1. Model: Stejný jako u MVC s tím, že se změny dat nikam nepropagují.
2. View: Zobrazuje tzv.provázaná data z ViewModelu. I zde je view zodpovědné za zachyceníakce jako u MVC.
3. View-Model: Manipuluje s modelem, změny dat se propagují díky data bindingu.

Provázání dat View s View-Model zde tedy svým způsobem nahradí Controller, nicméně implementace je trochu náročnější. Implementace obvykle probíhá pomocí rámce pro tzv. *data binding*, neboli datové provázání. U této konkrétní Aplikace pro prohlížení signálů by se jednalo o značné přírůstky kódu v případě použití MVVM.

2.6 Android Studio

Cílem práce bylo vytvořit nativní Android aplikaci. Nativní aplikací se myslí aplikace, která byla cíleně vytvořena pro jednu konkrétní platformu. Aplikace je z větší části psaná v jazyce Kotlin a její menší část v jazyce Java. Konkrétní důvody, srovnání a popis jazyků budou rozebrány v kapitole 2.7. Při vývoji aplikace byl zvolen Git, jako verzovací systém a vývojové prostředí Android studio.

Android Studio je vývojové prostředí založené na IntelliJ IDEA. Jedná se o vývojové prostředí s otevřeným zdrojovým kódem[4]. Umožňuje spravovat všechny potřebné části projektu. Na obrázku 5 lze vidět jak Android Studio vypadá, v levé části obrazovky najdeme způsob zobrazení souborů, podle výběru se poté zobrazuje struktura projektu, v této struktuře najdeme:

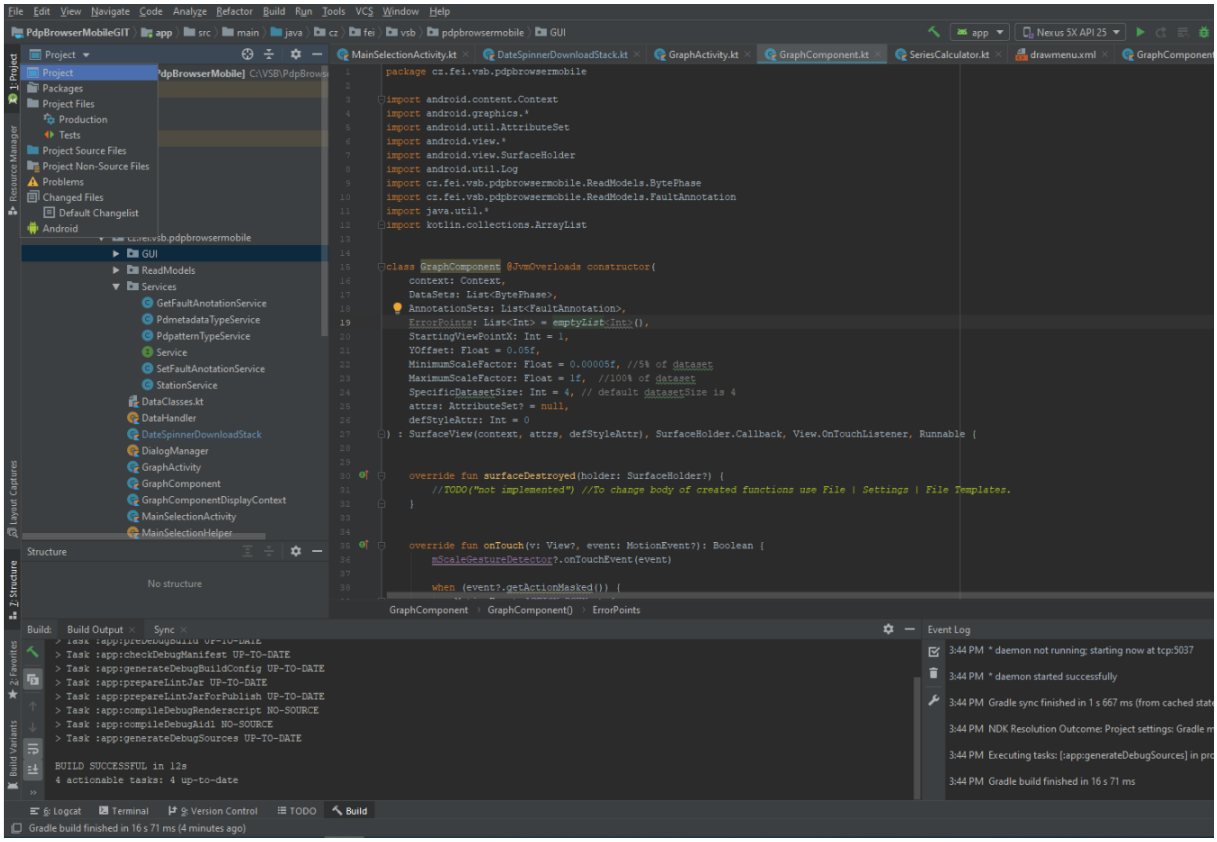
1. balíčky, jenž obsahují Java a Kotlin kód,
2. grafické prvky v podobě XML souborů,
3. používané knihovny,
4. tzv. *build.gradle* soubor, jenž obsahuje konfiguraci pro kompilaci, definuje verze SDK a závislosti,
5. tzv. *AndroidManifest.xml* soubor, který nesmí chybět v žádném projektu, protože v něm jsou údaje o samotné aplikaci, jako jsou aktivity a oprávnění, které si aplikace nárokuje.

Výhodou vývojového prostředí Android Studio je i editor pro tzv. *layout*. Jedná se o již několikrát zmíněný XML soubor, který obsahuje přesnou definici vzhledu konkrétní aktivity nebo konkrétního prvku. Prvkem může být například řádek v netriviálním seznamu položek.

Android Studio poskytuje celou řadu nástrojů pro ladění vyvíjené aplikace.

1. APK Analyzátor, umožňuje provést analýzu velikosti dat výsledné aplikace, resp. jejich částí.
2. Emulátor, součástí Android Studia je Android Virtual Device manager, což je nástroj, který umožňuje definovat si virtuální zařízení, na kterém lze aplikaci spustit bez nutnosti použití fyzického zařízení.
3. Profilovací nástroj pro měření využití hardware zařízení.

[4]Ještě před několika lety by stálo za to vyčlenit alespoň jednu kapitolu o přípravě vývojového prostředí. Jednalo se o docela náročný proces, zejména pro začínajícího programátora. Zvlášť se muselo například instalovat Android SDK, rozšíření do vývojového prostředí a provádět dodatečné nastavování. Vyskytovaly se časté problémy se samotnými balíčky. Dnes se současnou verzí AS je vše tak snadné, automatizované a integrované, že popis instalačního procesu, případně stahování, nemá smysl.



Obrázek 5: Struktura projektu

2.7 Kotlin a Java

Jak již zaznělo dříve, větší část Aplikace je napsána v jazyce Kotlin. Jedná se o jazyk, vyvinutý společností JetBrains. Je cíleně tvořen jako nástupce Javy, která existuje již zhruba 25 let, což je vlastně jeden z důvodů vzniku Kotlinu. Java si sebou totiž nese spoustu problémů z dob minulých a to zejména pokud se na danou problematiku díváme z pohledu vývoje pro platformu Android.

Výhoda Kotlinu je v tom, že je plně interoperabilní s Javou, což nám umožňuje mít nějaké části aplikace v Javě a jiné zase v Kotlinu. Za zmínku stojí fakt, že se Kotlin kompiluje do JVM byte-kódu, stejně jako Java. Konkrétně toho bylo v projektu využito v případě volání webových služeb, kde již dříve zmíněná knihovna kSOAP2 je obsluhována pomocí Javy. Interoperabilita tím byla ověřena v praxi a navíc bylo vhodnější při obsluze kSOAP2 vycházet z oficiální dokumentace, která je pouze pro implementaci v jazyce Java.

Cokoliv, co jde udělat v Javě, jde napsat v Kotlinu, nicméně mnohdy lépe a takřka vždy rychleji. Obecně by bylo vhodné podívat se i na některé konkrétní rozdíly Kotlinu oproti Javě.

- Podpora tzv. *null safe*. Kotlin je ve výchozím stavu pro všechny typy tzv. non-nullable. V případě, že se pokusíme do proměnné uložit `null`, vývojové prostředí nahlásí chybu již

v průběhu kompilace. Pro případ zpětné integrace s Java kódem máme možnost označit proměnnou jako nullable.

- Funkcionální část jazyka. Kotlin obsahuje například tzv. odložené vyhodnocování, lambda výrazy, přetížení operátorů a další pokročilé programátorské konstrukce.
- Datové typy. Kotlin je typovaný jazyk, nicméně není třeba typ uvádět, pokud nechceme, překladač jej pozná dle přiřazení.
- Datové třídy. Pokud je třída určena k držení dat, můžeme ji v podstatě napsat na jeden řádek v závislosti na počtu atributů. Netřeba psát tzv. getter a setter pro jednotlivé atributy.
- Odchytávání výjimek. V Kotlinu není třeba explicitně definovat a odchytávat výjimky, tak jak tomu je u Javy. Jinými slovy, nenastane situace, kdy jsme kompilátorem nuceni blokem `try catch` odchytávat výjimku.

Dalším důvodem pro upřednostnění jazyka Kotlin může být i oficiální oznámení Google z roku 2017, že oficiálním podporovaným jazykem je nyní Kotlin [8]. Na na závěr lze pozorovat porovnání ve výpisu 2.7.1 a 2.7.2, jakožto na důkaz o úspornosti. Vše co umí Java třída `Station`, umí i její Kotlin verze. V následujících výpisech kódu lze porovnat dvě třídy, které plní stjenou funkci. V jazyce Kotlin není třeba přetěžovat počítání HashCode, hloubkové porovnání, není třeba psát konstruktory pro inicializaci hodnot, není třeba psát tzv. „gettery“ a „settery“.

2.7.1 Porovnání - Java

```java
public final class Station {
    private String name;
    private int id;
    private float percentSize;

    public Station(String name, int id, float percentSize) {
        this.name = name;
        this.id= id;
        this.percentSize= percentSize;
    }

    public Station (String name, int id) {
        this.name = name;
        this.id= id;
        this.percentSize= 1.8f;
    }

    @Override
    public String toString() {
        return "Station(" + "name='" + name + '\'' + ", id=" + id + ",
            percentSize=" + percentSize + ')';
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        Station station= (Station) o;

        if (id != station.id) return false;
        if (Float.compare(station.percentSize, percentSize) != 0) return false;
        return name != null ? name.equals(station.name) : station.name == null;
    }

    @Override
    public int hashCode() {
```

```
37          int result = name != null ? name.hashCode() : 0;
38          result = 31 * result + id;
39          result = 31 * result + (percentSize!= +0.0f ? Float.floatToIntBits(
                percentSize) : 0);
40          return result;
41      }
42
43
44      public String getName() {
45          return name;
46      }
47
48      public void setName(String name) {
49          this.name = name;
50      }
51
52      public int getId() {
53          return id;
54      }
55
56      public void setId(int id) {
57          this.id= id;
58      }
59
60      public float getPercentSize() {
61          return percentSize;
62      }
63
64      public void setPercentSize(float percentSize) {
65          this.percentSize= percentSize;
66      }
67
68  }
```

Výpis 1: Ukázka třídy v jazyce Java

2.7.2 Porovnání - Kotlin

```kotlin
data class Station(
    var name: String,
    var id: Int,
    var percentSize: Float = 1.0f
)
```

Výpis 2: Ukázka třídy v jazyce Kotlin

3 Návrh aplikace

Tato část diplomové práce se bude věnovat návrhu mobilní aplikace jako takové. Bude popsána datová vrstva, jejíž součástí bude popis webových služeb. S návrhem aplikace souvisí i návrh a implementace android komponenty pro vykreslení grafu, jenž je také součástí následujících kapitol. V poslední řadě bude popsán návrh aktivit pro aplikaci.

3.1 Datová vrstva – popis webové služby

Pro účely získávání dat je publikována WCF služba. Aplikace využívá metody webové služby, které jsou popsány v tabulce 1. Tyto služby jsou poté zpracovány konkrétními třídami pro zpracování, viz. kapitola 3.2.1.

Tabulka 1: Metody webové služby

Název metody	**Parametry**	**Návratová hodnota**
`GetStations`	-	Seznam dostupných stanic. [3.1.1]
`GetMetadataFromTo`	1. `from` 2. `to`	Seznam metadat pro konkrétní měření. [3.1.2]
`GetPdpattern`	1. `idMeasurement` 2. `phase`	Metadata pro konkrétní fázi měření včetně dat ke zobrazení. [3.1.3]
`GetFaultAnnotation`	1. `idMeasurement` 2. `phase`	Hodnota anotace. [3.1.4]
`SetFaultAnnotation`	1. `idMeasurement` 2. `phase` 3. `value`	- [3.1.5]

Přes tyto metody jsou dostupná potřebná data a měření. Pro samotnou obsluhu webové služby, resp. konkrétních metod, byla vybrána knihovna třetí strany kSOAP2-android[5]. Jedná se o Simple Object Access Protocol klienta pro realizaci volání zmíněných metod dostupných v rámci webové služby. Jednotlivé funkce jsou rozepsány v následujících kapitolách.

[5] https://simpligility.github.io/ksoap2-android/

3.1.1 GetStations

Metoda nemá žádné vstupní parametry. Slouží k získání seznamu stanic. Získané stanice se ukládají jako seznam objektů třídy `StationType`. Každý tento objekt symbolizuje konkrétní stanici. V rámci každého objektu třídy `StationType` jsou dostupná následující data.

Tabulka 2: Parametry objektu třídy `StationType`

Název	**Datový typ**	**Popis**
`IPadd`	`int`	IP adresa stanice.
`IdStation`	`String`	Unikátní identifikátor stanice.
`Name`	`String`	Název stanice.
`X`	`float`	GPS souřadnice stanice.
`Y`	`float`	GPS souřadnice stanice.
`cislo_vedeni`	`String`	Číslo vedení
`prurez`	`String`	Velikost průžezu vedení.
`rozvadec`	`String`	Typ rozvaděče.
`rtu`	`String`	Jednotka dálkového přenosu.
`technicke_misto`	`String`	Technické místo
`typ_vodice`	`String`	Označení typu vodiče.
`umisteni`	`String`	Poloha stanice (název místa).
`usek`	`String`	Název úseku vedení na kterém stanice měří.

3.1.2 GetMetadataFromTo

Metoda slouží k získání seznamu metadat. Ta se ukládají v podobě objektů třídy `PdmetadataType`. Vstupními parametry jsou dva textové řetězce, viz. tabulka 3.

Tabulka 3: Vstupní parametry `GetMetadataFromTo`

Název	Datový typ	Popis
`from`	`String`	Časové razítko.
`to`	`String`	Časové razítko.

Jedná se o časová razítka ve formátu YYYYMMDDHHMMSS. V rámci výsledného listu objektů třídy `PdmetadataType` jsou dostupná data, která jsou uvedena v tabulce 4. Tato metadata nesou doplňkové informace k jednotlivým měřením.

3.1.3 GetPdpattern

Metoda slouží pro získání dat ke konkrétní fázi konkrétního měření. Vstupní parametry popsány v tabulce 5.

Výsledkem jednoho konkrétního volání je objekt třídy `PdpatternType`. Tento objekt obsahuje data popsána v tabulce 6.

Tabulka 4: Parametry objektu třídy `PdmetadataType`

Název	Datový typ	Popis
`Batt`	`String`	Stav baterie.
`ExtTemp1`	`String`	Venkovní teplota.
`ExtTemp2`	`String`	Venkovní teplota.
`Humidity1`	`String`	Vlhkost.
`Humidity2`	`String`	Vlhkost.
`IdMeasurement`	`int`	Identifikátor konkrétního měření, ke kterému patří metadata.
`IdStation`	`int`	Identifikátor stanice, ke které patří metadata.
`IndicatorFftU1`	`String`	Indikátor FftU1.
`IndicatorFftU2`	`String`	Indikátor FftU2.
`IndicatorFftU3`	`String`	Indikátor FftU3.
`IntTemp1`	`String`	Teplota vnitřního prostředí měřící stanice.
`Pwr`	`String`	Tlak.
`Status`	`String`	Status.
`Timestamp`	`String`	Časové razítrko měření.
`Wind`	`String`	Síla větru.

Tabulka 5: Vstupní parametry `GetFaultAnnotation`

Název	Datový typ	Popis
`idMeasurement`	`int`	Identifikátor měření.
`phase`	`int`	Číslo označující konkrétní fázi, může Nabývat hodnot v rozmezí 1 až 4.

Tabulka 6: Parametry objektu třídy `PdpatternType`

Název	Datový typ	Popis
`dataField`	`String`	Popis.
`idMeasurementField`	`int`	Popis měření.
`phaseField`	`int`	Popis fáze.
`samplingRateField`	`Integer`	Popis vzorkovací frekvence.
`voltageRangeField`	`Integer`	Popis rozsahu napětí.
`data`	`String`	Data ve formátu Base64[9], slouží k získání konkrétních hodnot, které jsou určeny k zobrazení.
`idMeasurement`	`int`	Identifikátor měření
`phase`	`int`	Číselné označené fáze.
`samplingRate`	`Integer`	Vzorkovací frekvence.
`voltageRange`	`Integer`	Rozsah napětí.

3.1.4 GetFaultAnnotation

Metoda slouží k získání anotace pro konkrétní fázi konkrétního měření. Anotace je číslo symbolizující stav konkrétní fáze. Je nastavováno ručně operátorem a to metodou `SetFaultAnnotation`, viz. kapitola 3.1.5.

- -1 - Žádná anotace.
- 0 - Bez chyby.
- 1 - Nepotvrzený kontakt stromu s krytým vodičem - kontakt uzemněného stromu.
- 2 - Nepotvrzený kontakt větve stromu s krytým vodičem - není kontakt větve se zemí.
- 3 - Porucha/přerušení.
- 4 - Sekundární porucha, porušení izolačního systému předchozí poruchou.
- 5 - Potvrzený kontakt stromu s krytým vodičem - kontakt uzemněného stromu.
- 6 - Potvrzený kontakt větve stromu s krytým vodičem - není kontakt větve se zemí.

Vstupní parametry popsány v tabulce 7 .

Tabulka 7: Vstupní parametry `GetFaultAnnotation`

Název	**Datový typ**	**Popis**
`idMeasurement`	`int`	Identifikátor měření.
`phase`	`int`	Číslo označující konkrétní fázi, může Nabývat hodnot v rozmezí 1 až 4.

Metoda vrací textový řetězec s jednou ze zmíněných hodnot -1 až 6.

3.1.5 SetFaultAnnotation

Metoda slouží k nastavení anotace pro konkrétní fázi konkrétního měření.

Má tyto vstupní parametry:

Tabulka 8: Vstupní parametry `SettFaultAnnotation`

Název	**Datový typ**	**Popis**
`idMeasurement`	`int`	Identifikátor měření.
`phase`	`int`	Číslo označující konkrétní fázi, může Nabývat hodnot v rozmezí 1 až 4.
`value`	`int`	Hodnota pro uložení, může Nabývat hodnot v rozmezí -1 až 6.

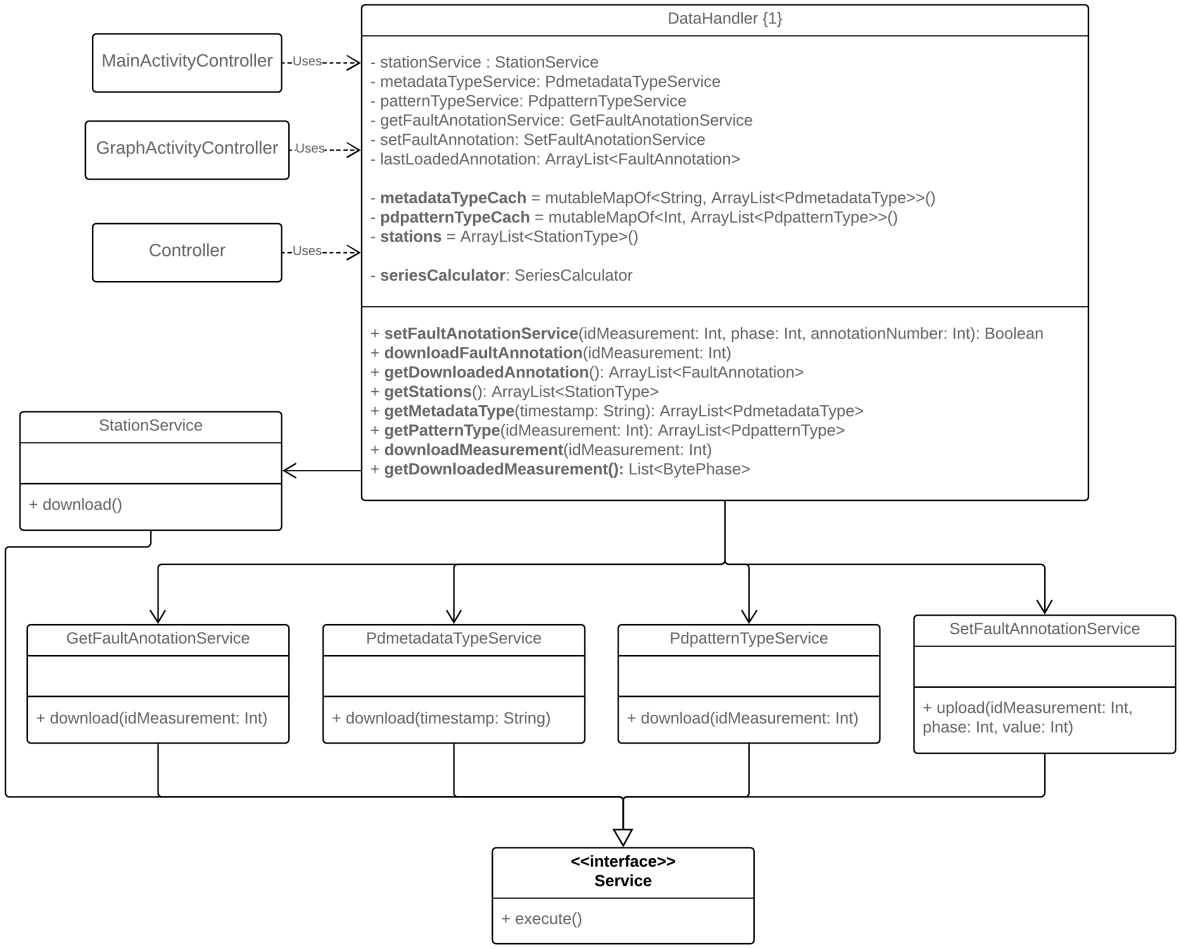
3.2 Datová vrstva - kSOAP2, DataHandler

Celá datová vrstva stojí na návrhovém vzoru fasáda. Diagram konkrétní implementace lze prohlédnout na obrázku 6. Třída `DataHandler` je označena v kódu klíčovým slovem `Object` a tudíž se jedná o singleton. DataHandler je použit napříč aplikací jako fasáda, přes kterou jsou snadno dostupná všechna data, která jsou načítána přes webové služby. V rámci jakékoliv aktivity se můžeme tedy jedním příkazem doptat na potřebná data.

Používání vzoru Jedináček je v jazyce Kotlin snadná a bezpečná technika. Implementace jedináčka nahrazuje statické třídy v Javě. Není třeba psát žádný dodatečný kód, tento způsob implementace vzoru Jedináček pomocí klíčového slova object nám garantuje bezpečný a efektivní způsob implementace tohoto vzoru pomocí jediného klíčového slova [10].

Díky tomuto je ve spojení s fasádou implementován jednoduchý princip ukládání do paměti, díky kterému se potřebná data a metadata stahují pouze jednou a jsou dostupná v rámci celé aplikace. Tato stažená data jsou uložena pouze v dočasné paměti a po vypnutí aplikace se ztrácí, nicméně je zamýšleno rozšíření, které umožní ukládat stáhnuté data do lokální SQLite databáze. Ukládání do paměti je možné, protože se data do databáze ukládají a nemodifikují se. Ukládání se netýká pouze anotací, protože ty se měnit mohou.

`DataHandler` používají třídy `MainActivityController`, `GraphActivityController` a `Controller`, tyto uvedené třídy jsou popsány v kapitole 3.3 a kapitole 3.4.



Obrázek 6: Datová vrstva

3.2.1 DataHandler

Jak již zaznělo v úvodu, `DataHandler` se chová jako fasáda. Při jeho implementaci je využit návrhový vzor jedináček a jeho snadná a bezpečná implementace v jazyce Kotlin. Součástí třídy `DataHandler` jsou objekty pro obsluhu jednotlivých metod webové služby. Tyto objekty mají implementované metody `download` nebo `upload`. Jedna nebo druhá ze zmíněných metod slouží k nastavení parametrů a věcí s tím spojených. Samotné spuštění akce (volání webové služby) má na starosti přetížená metoda `execute`, kterou má prostřednictvím rozhraní `IService` každá třída pro jednotlivé metody. Většina metod objektu třídy DataHandler je mapovaných prostřednictvím zmíněných tříd přímo na metody webové služby, viz. tabulka 9. Jedná se o objekty tříd:

1. `GetFaultAnnotationService`,
2. `PdmetadataTypeService`,
3. `PdpatternTypeService`,
4. `SetFaultAnnotationService`,
5. `StationService`.

Před tím, než jsou vysílány požadavky na načtení dat se v objektech třídy `DataHandler` hledá dle klíčů, zda náhodou data nejsou již k dispozici. K tomuto slouží objekty `metadataTypeCach`, `pdpatternTypeCach` a `stations`. Tento jednoduchý systém cache zamezí zbytečnému znovustahování stejných dat, alespoň v rámci jednoho běhu aplikace. V budoucnu je naplánována impementace offline cache v podobě SQLite databáze pro podporu offline režimu.

V rámci třídy `DataHandler` se můžeme dostat pomocí metod v tabulce 9 ke všem potřebným datům. Čísla v hranatých závorkách na konci významu odkazuje na příslušnou kapitolu s detailním popisem.

3.2.2 SeriesCalculator

Tato jednoduchá třída nepatří do datové vrstvy jako takové. Nepodílí se na komunikaci s webovými službami. Je zde zmíněna, protože má za úkol prosté převedení dat měření z formátu Base64 na objekty třídy `ByteArray` a jejich následné uchování. Výsledné objekty se poté použijí pro vytvoření seznamu objektů třídy `BytePhase`. Tento seznam objektů pro konkrétní měření se použije jako vstup do komponenty `GraphComponent`.

Třída `BytePhase` obsahuje výsledná data ke zobrazení a údaj o tom, o kterou fázi se jedná. Celý proces převodu dat se spouští paralelně pro všechny čtyři fáze zároveň. Výsledný seznam objektů třídy `BytePhase` slouží jako vstup pro třídu `GraphComponent`. Více informací o komponentě v kapitole 3.3.3, k dispozici je také diagram 9.

Tabulka 9: Metody třídy `DataHandler`

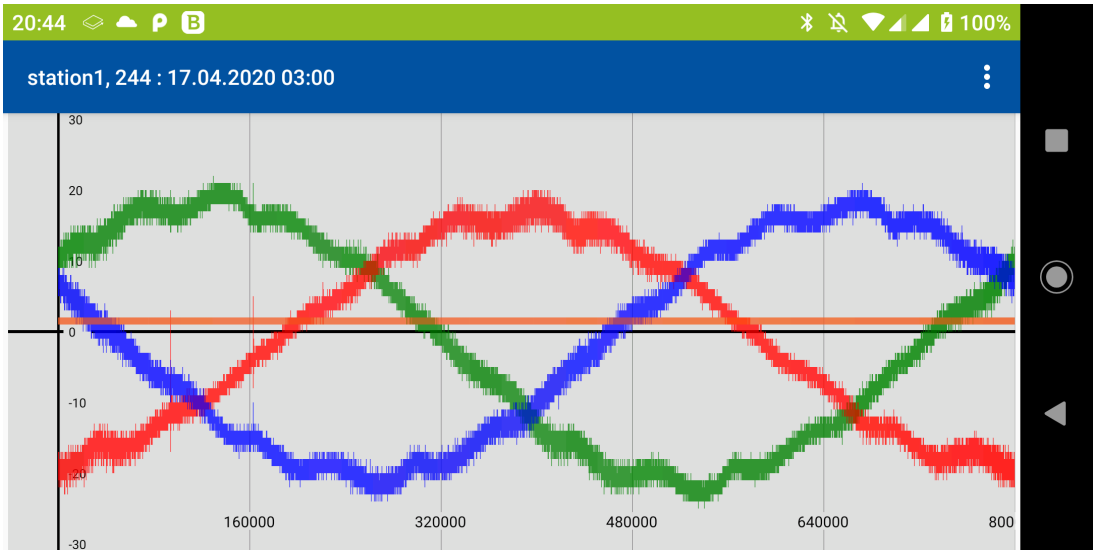
Název metody	**Vstupní parametry**	**Význam**
`setFaultAnotationService`	1. Id měření 2. Číslo konkrétní fáze 3. Hodnota fáze [3.1.4]	Odešle požadavek pro přiřazení anotace [3.1.5].
`downloadFaultAnnotation`	1. Id měření	Stáhne anotace pro všechny fáze konkrétní měření [3.1.4].
`getDownloadedAnnotation`	-	Vrátí stažená data v podobě seznam objektů anotací [3.1.4].
`getStations`	1. Id měření	Stáhne a vrátí seznam stanic, pokud jsou data již stažena, pouze vrátí. [3.1.1]
`getMetadataType`	1. Časové razítko	Stáhne a vrátí seznam objektů s metadaty, pokud jsou data již stažena, pouze vrátí. [3.1.2].
`getPatternType`	1. Id měření	Stáhne a vrátí seznam objektů s daty, nese nezpracované data s měřením v base64 formátu a další doplňkové data k měření [3.1.3]. Pokud jsou data již stažena, pouze je vrátí.
`downloadMeasurement`	1. Id měření	Vyšle požadavek ke stažení a převedení dat [3.2.2].
`getDownloadedMeasurement`	-	Vrátí stažené a přepočítané měření [3.2.2].

3.3 Android komponenta GraphComponent

Tato komponenta je jednou z klíčových částí aplikace, musí zvládnout pracovat se čtyřmi signály o celkovém počtu 800 000 jednotlivých měření pro každý signál. Důvod k vytvoření vlastní komponenty vznikl při vývoji této aplikace v rámci semestrálního projektu. Při tomto prvním pokusu se ukázalo, že doposud neexistuje žádná připravená Android komponenta, která by sloužila k zobrazení grafu, podobně jako tomu je například u komponenty pro výběr data. Existují sice knihovny třetích stran, jako je například `GraphView`[6], ukázalo se, že tyto knihovny nezvládají práci s tak rozsáhlými daty. Výběr této nebo obdobné knihovny třetí strany by byl použitelný pouze v případě, že bychom museli zobrazovat mnohem menší sadu dat, nebo kdybychom ji mohli nějak zjednodušit. Konkrétní problém je v optimalizaci podobných řešení, která obecně jsou příliš generická a nezvládají zobrazovat tak velké množství dat bez zjednodušení, nebo ztráty detailů signálu, což je jedna z klíčových vlastností tohoto projektu.

3.3.1 Popis komponenty GraphComponent

V této podkapitole bude čtenář pro lepší představu seznámen se vzhledem komponenty, jejími funkcemi a základními pojmy, které budou použity v podkapitolách následujících. Na obrázku 7 lze vidět základní zobrazení komponenty. Tedy zobrazení po otevření měření. Uživatel vidí průběh všech fází v maximálním možném oddálení tzn. vidí celý průběh měření.



Obrázek 7: Základní zobrazení komponenty

Komponenta umí vykreslit i chybové body. Jedná se o možnost poskytnout komponentě seznam bodů ke zvýraznění. Těmto bodům se říká chybové body. Ukázka vykreslení chybového bodu je na obrázku 16. Cybové body určuje klasifikátor, který se stále vyvíjí a není součástí této práce. Zatím není k dispozici webová služba, která by data poskytovala. V budoucnu se s ní

[6]Nalezneme ho jako součást balíčku com.jjoe64:graphview:4.2.2.

počítá, aplikace si proto sama generuje chybové body. Zobrazené chyby, resp. chybové body, jenž se objeví na případných snímcích, slouží pouze pro demonstraci funkčnosti a jejich identifikace probíhá na základě nepřesné demonstrativní metody v rámci jedné z aktivit. Popis tohoto řešení v kapitole 3.4.2.

Měření je zobrazeno v podobě grafu. Osa X představuje časový průběh měření. Vždy je k dispozici 800 000 jednotlivých měření, resp. hodnot. Tyto hodnoty jsou zaznačeny na ose Y. Mohou nabývat hodnot v rozmezí -128 až 127[7].

V ose X je pro usnadnění orientace zobrazena mřížka. Zároveň s mřížkou uživatel vidí popisky hodnot, které se překreslují zároveň s mřížkou. Uživatel tak po přiblížení ví, v jaké části grafu se nachází.

Počátek osy X nezačíná od prvního pixelu displeje, je o určitou hodnotu posunut. Tato hodnota ve výchozím nastavení komponenty odpovídá 5% šířky displeje. Říkáme ji posun osy Y.

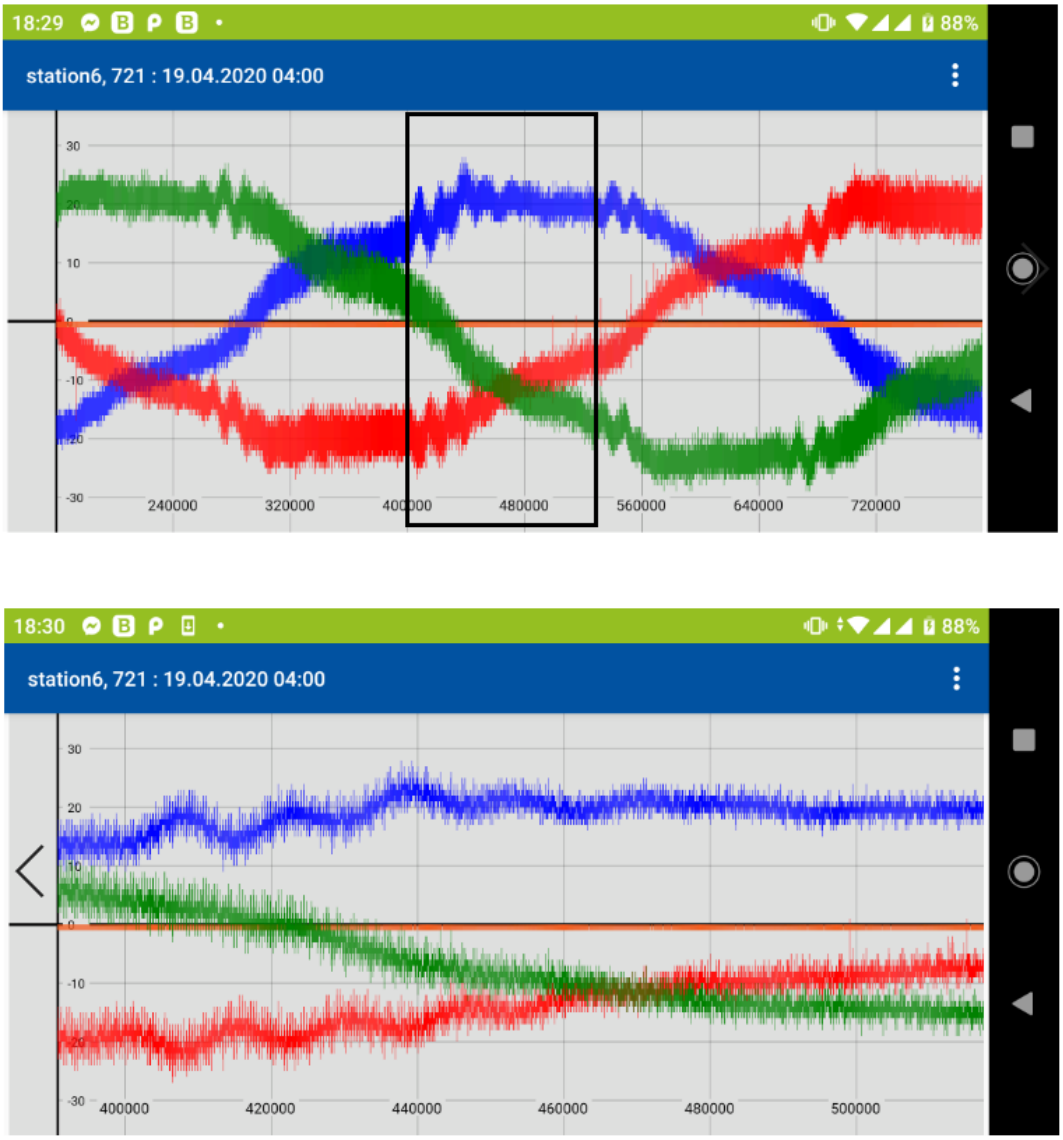
Uživatel má možnost se přiblížit a libovolně se pohybovat v rámci měření doprava a doleva. Maximální přiblížení odpovídá 0.005% bodům poskytnutých datových sad ke zobrazení[8]. V případě, kdy je k dispozici 800 000 měření, to odpovídá 40-ti hodnotám, které budou na displeji viditelné v maximálním přiblížení. Maximální oddálení ve výchozí hodnotě odpovídá zobrazení 100% datových sad, je tedy viditelné vše. Obě hodnoty, tedy maximální přiblížení a oddálení, lze v rámci konstruktoru komponenty upravovat. Pro pohyb v grafu (tj. přiblížené, oddálení a posun) slouží konvenční gesta známá i z ostatních Android aplikacích. Na obrázku 8 lze v prvním snímku displeje telefonu vidět zvýrazněnou oblast, na kterou se uživatel chce přiblížit. Toto zvýraznění v podobě rámečku slouží pouze k demonstraci, nejedná se o žádnou součást aplikace. Pomocí gesta pro přiblížení a případného použití posunu se do cílené lokace dostane, viz. druhý snímek obrazovky na obrázku 8.

Při přiblížení se mění pozice okna pro vykreslení. To je oblast v měření, která je zobrazená uživateli přes celý displej přesně tak, jak je to vysvětleno na obrázku 8. Okno pro vykreslení vykreslí graf podle tzv. počáteční indexu, což je bod v souřadnicích měření, který identifikuje polohu okna pro vykreslení. Je to tedy první viditelný bod z měření k zobrazení.

Pro další ovládání komponenty, mimo gesta pro pohyb v grafu, jsou připraveny metody pro skrytí a odkrytí fáze a reset měřítka do původní podoby. Původní podobou se myslí výchozí vzhled při otevření měření, při kterém jsou viditelné všechny body měření. Tyto metody jsou potom využity v rámci menu aktivity, která komponentu používá.

[7] U bezproblémových měření se obvykle drží v rozmezí -30 až 30

[8] Datovou sadou se myslí seznam objektů BytePhase, které nesou data ke zobrazení a informaci o tom, o jakou fázi se jedná

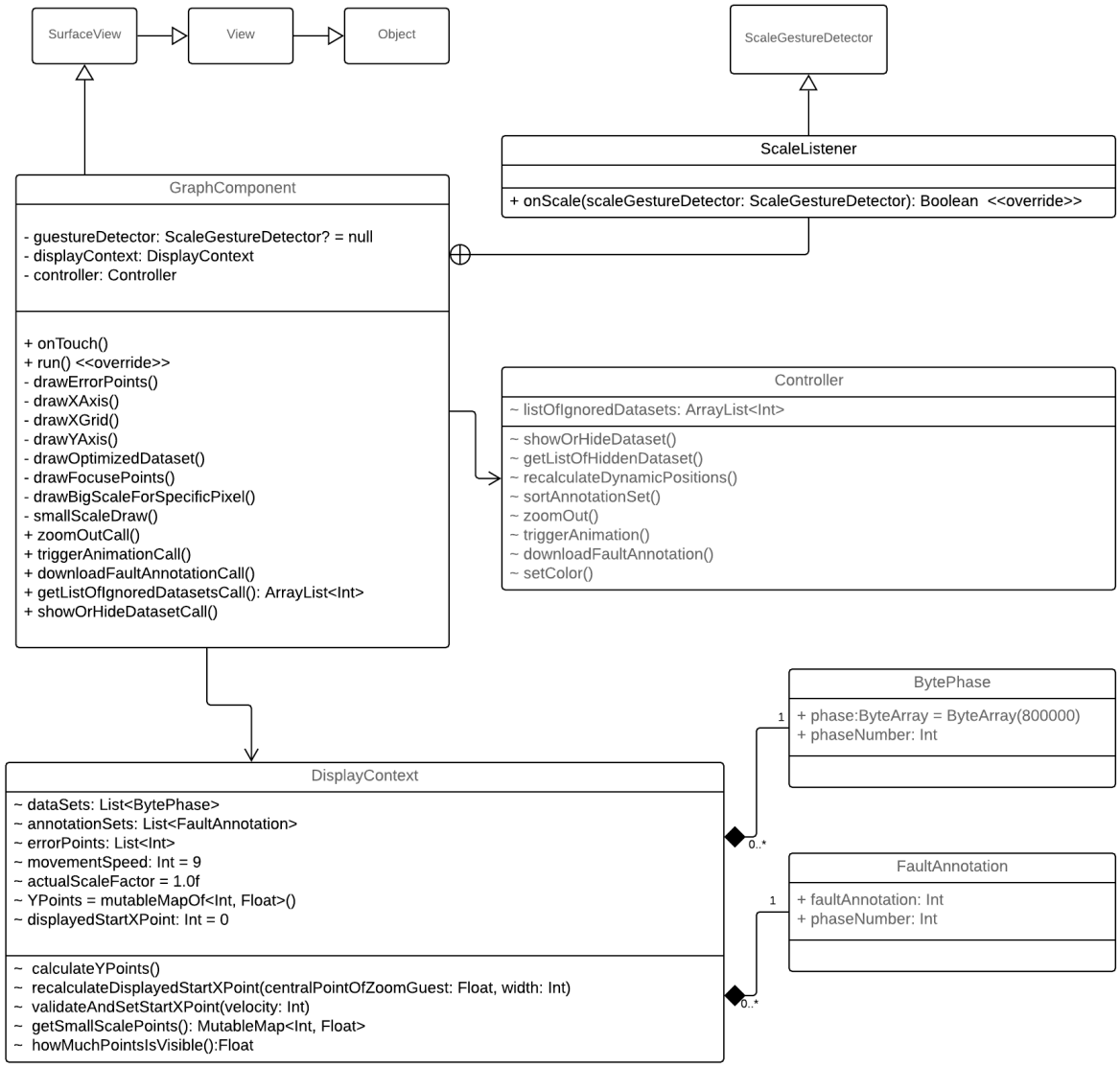


Obrázek 8: Proces přiblížení

3.3.2 Architektura komponenty GraphComponent

Na obrázku 9 lze vidět UML třídní diagram návrhu vlastní komponenty. Tento diagram byl pro přehlednost zjednodušen. Vypuštěny byly vstupní parametry některých funkcí a některé privátní funkce a proměnné. Na tomto diagramu lze pozorovat tři klíčové třídy, tedy `GraphComponent` `DisplayContext` a `Controller`, které budou popsány v následujících podkapitolách.

Méně důležité třídy jsou `BytePhase` a `FaultAnnotation`, které slouží pouze jako třídy pro přenos dat.



Obrázek 9: Základní zobrazení komponenty

3.3.3 Třída `GraphComponent`

Jedná se o nejzásadnější třídu. Zodpovídá za vykreslování a zpracovávání vstupů. Obsahuje výpočty související s grafikou, tedy vykreslováním. Přes tuto třídu se komponenta inicializuje v rámci aktivity `GraphActivity`. Použití komponenty je jednoduché, jako příklad nechť poslouží výpis kódu 3.

```
1  // inicializace objektu
2  graphComponent = GraphComponent(
3     this,
4     DataHandler.getDownloadedMeasurement(),
5     DataHandler.getDownloadedAnnotation(),
6     generateDemoErrorPoints()
7  )
8
9  // Vytvoření objektu s parametry, který obsahuje nastavení pro layout
10 var params : ConstraintLayout.LayoutParams = ConstraintLayout.LayoutParams(
11    CoordinatorLayout.LayoutParams.MATCH_PARENT,
12    CoordinatorLayout.LayoutParams.WRAP_CONTENT
13 )
14
15 // Přířazení vnějšího okraje
16 params.setMargins(10,10,10,10)
17
18 // Předání objektu s parametry
19 graphComponent.layoutParams = params
20
21 // Nastavení vnitřního okraje přímo na komponentě
22 graphComponent.setPadding(10,10,10,10)
23
24 // Přidání komponenty na existující prvek v XML dokumentu aktivity
25 graph_layout.addView(graphComponent)
```

Výpis 3: Ukázka použití komponenty GraphComponent

Na řádku 2 lze vidět objekt `graphComponent`, ten je objektem třídy `GraphComponent`. Pro správné použití je třeba konstruktoru předat kontext a sadu dat měření k zobrazení. To se děje na řádku 3 a 4. Data pro měření jsou ve chvíli použití již zpracovaná. Zpracování se v tomto konkrétním případě děje v předchozí aktivitě po vybrání konkrétního měření. Na řádku 5 a 6 lze vidět přiřazení dalších nepovinných položek. Jedná se o seznam anotací a seznam chybových bodů.

Konstruktor nabízí možnost použití různých parametrů. Nepovinné parametry jsou nastaveny v případě neposkytnutí na výchozí hodnoty. Kompletní popis parametrů je dostupný v tabulce 10. Výchozí hodnoty jsou uvedeny ve sloupci „Výchozí hodnota“. Pokud je uvedeno klíčové slovo „Není“, parametr je povinný.

Tabulka 10: Konstruktor `GraphComponent`

Název metody	**Datový typ**	**Výchozí hodnota**	**Poznámky**
`context`	`Context`	Není	Vždy je nutno předat kontext aktivity.
`dataSets`	`List<BytePhase>`	Není	Datový set, vždy je nutno předat.
`annotationSets`	`List<FaultAnnotation>`	Prázdný list	Anotace fází.
`errorPoints`	`List<Int>`	Prázdný list	Body ve kterých byla identifikována chyba.
`startingViewPointX`	`Int`	1	Pozice v grafu od kterého jsou data viditelná.
`yOffset`	`Float`	0.05f	Posun osy Y, udáván v procentech ku šířce displeje.
`minimumScaleFactor`	`Float`	0.00005f	Maximální přiblížení v procentech ku datovému setu.
`maximumScaleFactor`	`Float`	1f	Maximální oddálení v procentech ku datovému setu.

V inicializačním bloku uvnitř třídy `GraphComponent` se provede několik důležitých kroků pro správnou funkci komponenty. Jedná se o:

- inicializaci všech potřebných vnitřních proměnných a objektů,
- registraci posluchačů pro zpracování dotyků,
- vytvoření objektu třídy `Controller`,
- vytvoření objektu třídy `DisplayContext` a inicializace jeho dat,
- spuštění výpočtů souvisejících s vykreslováním, tzn. přepočet souřadnic, pro osu X a osu Y a také vytvoření mapy pro body osy Y (viz. podkapitola 3.3.4).

3.3.4 Mapa bodů osy Y

Mapa bodů v ose Y slouží k zamezení opakujících se výpočtů. Vzhledem k tomu, že se nemění měřítko osy Y, byl zvolen tento způsob určení souřadnice na displeji pro konkrétní hodnotu. Vypočítaná mapa bodů Y se na základě podoby zobrazovaných dat dynamicky přizpůsobí tak, aby graf vždy zabíral 80% displeje a po celou dobu běhu aplikace při vykreslování jednoho

konkrétního měření. Tak se šetří výkon a aplikace celkově nepůsobí matoucím dojmem, protože kvůli povaze měření je pro uživatele vhodnější, když dochází k posunu pouze v ose X.

Mapa bodů v ose Y je zaznačena na obrázku 9 v rámci třídy `DisplayContext`, jedná se o mapu `YPoints`, ta mapuje hodnoty osy Y, což jsou v tomto případě celá čísla, na hodnoty typu Float, což jsou souřadnice na displeji. Jedná se o jakousi obdobu slovníku v jazyce kotlin. Metoda `calculateYPoints` spouští výpočet.

Výpočet je sám o sobě lehký. Při výpočtu se zjistí, jaká maximální a minimální hodnota se v měření vyskytuje. Potom se každému celému číslu v tomto rozmezí přiřadí konkrétní souřadnice na displeji v ose Y na základě toho, kolik pixelů je celkem k dispozici.

Dalšími metodami dostupnými v rámci třídy jsou veřejné metody s příponou `call`. Tyto metody slouží pro zachycení požadavku z aktivity a pouze volají metody na objektu třídy `Controller`, kde se nachází jejich implementace.

3.3.5 Třída `DisplayContext`

Další důležitá část komponenty je třída `DisplayContext`. Objekt této třídy si pamatuje veškeré důležité informace, které se týkají kontextu komponenty, to znamená například polohu v grafu, data k vykreslení a také operace související s abstraktními výpočty. Abstraktními výpočty se myslí kalkulace zodpovědné za správnost zmíněných údajů důležitých pro pozdější správné vykreslení, které má na starosti předchozí popsaná třída (tj. `GraphComponent`). Konkrétním příkladem může sloužit přepočet počátečního indexu po změně měřítka grafu. Pro připomenutí ještě jednou, počáteční index je bod v měření signálu, od kterého jsou data na displeji viditelná, tedy bod, který se mění v závislosti na poloze okna pro vykreslení.

3.3.6 Třída `Controller`

V této třídě se nachází implementace metod, které lze chápat jako řídící akce na komponentě, které nesouvisí se zpracováváním uživatelských gest. Objekt této třídy si pamatuje seznam skrytých fází v seznamu `listOfIgnoredDatasets`. Metoda `showOrHideDataset` implementuje přidání či odebrání fáze ze seznamu. Metoda `getListOfHiddenDataset` vrací seznam skrytých fází.

Metoda `recalculateDynamicPositions` vyvolá přepočítání pozice X na displeji pro osu Y a také přepočítá mapu bodů Y(viz. podkapitola 3.3.4). Ve finální podobě je tato metoda použita pouze jednou, nicméně v případě povolení změn rozložení obrazovky na horizontální a vertikální režim lze použít tuto metodu. Metoda `sortAnnotationSet` třídí anotace tak, aby v seznamu jejich pozice šla za sebou, čehož je posléze využito při výpisu. Anotace se totiž stahují paralelně a tak se může stát, že pořadí stažení anotace ve výsledném seznamu neodpovídá logickému pořadí.Metoda `zoomOut` obsahuje implementaci resetu měřítka a polohy vykreslovacího okna na výchozí hodnoty, tedy do podoby po otevření měření. Metoda `triggerAnimation` spouští překreslení komponenty. V metodě `downloadFaultAnnotation` dochází k zažádání o stažení

anotací a setřídění anotací pomocí avizované metody k tomu určené. Poslední metodou je metoda `setColor`, která na základě vstupu nastaví barvu pro vykreslování, čehož je využito při vykreslování každé konkrétní fáze.

3.3.7 Výpočet pohybu okna pro vykreslování

Pro připomenutí, uživatelské vstupy v rámci vykreslované plochy jsou dvě různá uživatelská gesta:

- gesto pro přiblížení a oddálení,
- gesto pro posun do strany.

Výsledky uživatelských vstupů se projevují na objektu třídy `DisplayContext`, který má na starosti případné výpočty související s pohybem a stavem okna, ve kterém probíhá vykreslování.

Pro zachycení klasického posuvného gesta je přetížena metoda `onTouch` ve třídě `GraphComponent`. Její vnitřní logika zachycuje všechny události týkající se dotyku displeje. Zpracovávají se v ní tyto základní události:

- Detekce doteku, registruje začátek sledování události,
- detekce pohybu, zachytí pohyb a pošle hodnotu hybnosti objektu třídy `DisplayContext`.

Při procesech týkajících se pohybu okna pro vykreslování se musí přepočítávat aktuální počáteční index a při přiblížení nebo oddálení i měřítko. Pozice aktuálního počátečního indexu je uchovávána v rámci objektu třídy `DisplayContext` v proměnné `displayedStartXPoint`. Jedná se tedy o pozici, kde okno začíná a od kterého později probíhá vykreslování. Po zachycení hybnosti a předání této hodnoty metodě `validateAndSetStartXPoint` dojde k přepočtu `displayedStartXPoint`, což je náš počáteční index. Hybnost je bezrozměrná hodnota, jejíž hodnota je určena na základě stylu pohybu uživatele. V potaz jsou brány faktory jako délka pohybu prstu a rychlost pohybu prstu. Tato hybnost nabývá hodnot v řádech tisícu až nižších desetitisíců. Její určení závisí plně na odchycené události. Pokud je pohyb veden směrem doleva, hodnoty jsou záporné. Znamená to, že uživatel si přeje pohyb doprava. Pokud je pohyb veden směrem doprava, hodnoty jsou kladné. Znamená to, že si přeje pohyb doleva. Hodnota hybnosti je použita pro výpočet v jednoduchém vzorci. Pseudokód výpočtu s naměřenou hybností vypadá následovně.

```
1 pocatecniIndex -= (hybnost*(meritko*koeficientRychlosti))
```

Výpis 4: Výpočet počátečního indexu při posunu

Konstanta `koeficientRychlosti` je uložena v konstantě `movementSpeed` a slouží k získání vyšší kontroly nad rychlostí pohybu. Z pseudokódu je pro přehlednost vynecháno ošetření hraničních hodnot, tedy posun mimo graf. Metoda `validateAndSetStartXPoint` totiž obsahuje

i validaci a korekci hodnoty. V případě, že se výsledná hodnota dostane mimo reálné možnosti vykreslování, které jsou reálně k dispozici, nedovolí aby výpočet proběhl. Naopak pokud se dostane hodnota před reálné data, tedy do záporných hodnot, nastaví nejmenší možnou pozici okna, tedy 1.

Pro zpracování gesta přiblížení a oddálení je implementována vnitřní třída `ScaleListener`, tato třída dědí od `SimpleOnScaleGestureListener` a přetěžuje funkci `onScale`, která na základě objektu třídy `ScaleGestureDetector`[9], vypočítá procentuální změnu velikosti měřítka po provedeném gestu přiblížení nebo oddálení na základě získané hodnoty. Tato hodnota odpovídá změně měřítka. Zmíněný objekt `ScaleGestureDetector` je vstupním parametrem funkce přetížené metody `onScale`.

Konkrétní výpočet je dělení aktuálního měřítka atributu `actualScaleFactor` (jehož hodnotu si pamatuje objekt třídy `DisplayContext`, viz. diagram 9), pomocí atributu `scaleFactor`, jenž je dostupný prostřednictvím objektu třídy
`ScaleGestureDetector` zmíněného v předchozím odstavci.

V tomto procesu je zastřešena jak minimální, tak maximální hodnota. Výchozí hodnoty jsou měnitelné pomocí konstruktoru. Výchozí minimum je 0.005%, tedy při maximálním přiblížené bude viditelných 40 bodů z měření. Výchozí maximum je 1, tedy 100%, při maximálním možném oddálení potom bude viditelných všech 800 000 bodů měření ve všech fázích.

Detekovaná změna měřítka zároveň spouští metodu `recalculateDisplayedStartXPoint`, která má na starosti korekci počátečního indexu po změnách měřítka. Operuje s centrem uživatelského gesta. Pomocí tohoto centra dojde k určení tzv. cíleného bodu, ten odpovídá reálnému bodu měření, na který se uživatel chce přiblížit nebo oddálit. Tento reálný bod nazvěme tedy cíleným bodem. Výpočet je popsán následujícím pseudokódem s komentářem. Pro přehlednost byl pseudokód zjednodušen a to včetně vynechání například kontrol hraničních situací, tedy situace kdy by se uživatel mohl pozicí okna pro vykreslení dostat mimo graf. Tomu je zabráněno korekcí, která zajistí posun počátečního indexu určitým směrem tak, aby celé okno zůstalo pouze na souřadnicích měření z dat, které jsou k dispozici.

```
1  //Výpočet nedostupných pixelů
2  pocetNepouzitelnychPixelu = posunOsyY*sirka
3  //posunOsyY: procentuální hodnota (zlomek)
4
5  //Výpočet optimalizovaného centrálního bodu
6  optimalizovanyCentralniBod = centrumGestaVOseX-pocetNepouzitelnychPixelu
7
8  pouzitePixely = (sirka-pocetNepouzitelnychPixelu)
9
10 /* Výpočet zlomku, který nabývá hodnot v~otevřeném intervalu 0 až 1 a~tím přesn
       ě identifikuje polohu na ose X pouze v~místě použitém pro vykreslování. */
```

[9] https://developer.android.com/reference/android/view/ScaleGestureDetector

```
11 poloha = optimalizovanyCentralniBod/pouzitePixely
12
13 //metoda howMuchPointsIsVisible slouží k~získání množství aktuálně viditelného
       počtu pixelů.
14 novaVelikostOkna = howMuchPointsIsVisible()
15
16 //Určení cíleného bodu
17 cilenyBod = (novaVelikostOkna*poloha)+pocatecniIndex
18
19 //Využití cíleného bodu při výpočtu počátečního indexu
20 pocatecniIndex = cilenyBod-((novaVelikostOkna/2))
```

Výpis 5: Výpočet pozice okna pro vykreslování

Zároveň s uživatelským vstupem, který může mít vliv na podobu grafu, se spouští i vykreslení.

3.3.8 Vykreslování komponenty

Vykreslování je řešeno smyčkou, ve které se překreslí komponenta v případě, že došlo, resp. dochází k uživatelským vstupům. Je implementovaná třída `runnable` a přetížená metoda `run`. Pro vykreslování je použito jedno vlákno `drawThread`. Celkové vykreslení komponenty má na starosti několik metod s prefixem *draw*, které se volají z přetížené metody `run` ve stejném pořadí, v jakém jsou uvedeny v následujícím výpisu.

1. `drawXAxis` – Vykresluje osu X.
2. `drawYAxis` - Vykresluje osu Y, a popisky osy.
3. `drawErrorPoints` - Vykreslí chybové body.
4. `drawXGrid` - Vykresluje mřížku osy X a její popisky.
5. `drawOptimizedDataset` - Tato metoda dále distribuuje vykreslování jednotlivých signálů za pomocí dvou následujících metod a to v závislosti na měřítku.

Metody `drawBigScaleForSpecificPixel` a `smallScaleDraw` nejsou ve výpisu uvedeny, neboť se nevolají přímo. Jejich volání zařizuje poslední zmíněná metoda ve výpise a tou je `drawOptimizedDataset`. Tato metoda totiž rozhoduje dále o tom, jak proběhne vykreslování samotné, ale sama o sobě nic nekreslí.

Komponenta řeší velký objem dat použitím dvojího typu vykreslování. Stále je třeba totiž vést v patrnost to, že je třeba zobrazit 800 000 měření pro každou ze čtyř fází, to je až 3 200 000 naměřených hodnot v jednu chvíli a to bez ztráty informací. Metoda nejdřív vypočítá počet dostupných pixelů na ploše pro kreslení v ose X. Potom porovná počet dostupných bodů s počtem bodů měření, které se mají vykreslit. Pokud je třeba vykreslit přesně nebo více

než dvakrát více bodů, než kolik je pixelů, je zvolen režim kreslení ve velkém měřítku. Pokud podmínka splněná není, vykresluje se v malém měřítku. Pro přehlednost lze chování metody `drawOptimizedDataset` popsat následujícím pseudokódem.

```
1  pocetViditelnychBodu = displayContext.howMuchPointsIsVisible
2  pocetDostupnychPixelu = sirka-(sirka*displayContext.posunOsyY)
3
4  if (pocetViditelnychBodu>=pocetDostupnychPixelu*2){//Kreslení ve velkém měřítku
5
6  //Math.ceil - Celočíselné dělení zaokrouhlené nahoru
7  pocetMereniProJedenPixel = Math.ceil(pocetDostupnychPixelu/pocetViditelnychBodu
       )
8
9  /* Vykreslování začne ve všech případech od počátečního indexu. */
10 procesovanyIndex = displayContext.displayedStartXPoint
11
12 for (i in 1..pocetDostupnychPixelu) {
13 /* xPoziceNaDispleji je vypočítána v hodnotě float, aby nedocházelo k viditelné
       mu skákání po pixelech. */
14 xPoziceNaDispleji = displayContext.getXPositionInFrame(procesovanyIndex)
15
16 /* Iterátor zajistí vykreslení čárky vgrafu pro každou konkrétní fázi v měření
       */
17 iterator = (0..displayContext.dataSets.size-1).iterator()
18 iterator.forEach {
19     controller.setColor(displayContext.dataSets[it].phaseNumber)
20     drawBigScaleForSpecificPixel(xPoziceNaDispleji, procesovanyIndex,
           pocetMereniProJedenPixel)
21 }
22
23 procesovanyIndex += pocetMereniProJedenPixel
24     }
25 }
26 else{ //Kreslení v malém měřítku
27     pocatecniIndex = displayContext.displayedStartXPoint
28     koncovyIndex = displayContext.validateEndSliceData(pocetViditelnychBodu+
           pocatecniIndex)
29
30     listDat = List
```

```
31      /* Průchod všemi datovými sadami k vykreslení a přidání jejich konkrétních
            částí k vykreslení do listuDat */
32      for(x in displayContext.dataSets){
33          data = x.faze.sliceArray(pocatecniIndex..koncovyIndex)
34          listDat.add(data)
35      }
36
37      xMapa = displayContext.getSmallScalePoints(width)
38      yMapa = displayContext.YPoints
39
40      /* Iterátor zajistí průchod fázemi a na každou fázi zavolá její vykreslení.
             Protože je bodů méně než pixelů, není třeba kreslit po pixelech jako v
            předchozím případě. */
41      val iterator = (0..displayContext.dataSets.size-1).iterator()
42      iterator.forEach {
43          smallScaleDraw(xMapa, yMapa, listDat)
44      }
45  }
```

Výpis 6: Chování metody drawOptimizedDataset

Režim kreslení ve velkém měřítku probíhá postupným voláním metody `drawBigScaleForSpecificPixel`. Tato metoda je volána tolikrát, kolik je k dispozici pixelů v ose X. Přesně jak tomu je v předchozím výpisu. Pro pixel je určen optimální počet měření na základě počtu měření ke zobrazení a počtu dostupných pixelů. Pro každý pixel je na základě počtu měření pro jeden pixel vyjmut specifický úsek dat z konkrétní datové sady. Datovou sadou se myslí měření. Na tomto specifickém úseku dat je poté nalezena maximální a minimální hodnota. Tyto dva body jsou spojeny čarou. Tento přístup zajistí, že uživatel vizuálně nepřijde o žádné body. Přesné souřadnice v ose Y pro všechny body v ose Y jsou dostupné v již vypočítané mapě `YPoints` prostřednictvím objektu `displayContext`. Souřadnice na displeji v ose X je poskytnuta předchozí metodou. Chování metody `drawBigScaleForSpecificPixel` nechť je popsáno následujícím pseudokódem.

```
1   /* vysekDatFazeUrceneProPixel - výsek dat pro konkrétní pixel, výpočet pro př
        ehlednost vynechán */
2   min = vysekDatFazeUrceneProPixel.min()
3   max = vysekDatFazeUrceneProPixel.max()
4
5   /* Pro přehlednost vynechány korekce v~podobě přičítání posunů. */
6   drawLine(
7       poziceXProProcesovanyIndex, yMapa[max]),
```

```
8       poziceXProProcesovanyIndex, yMapa[min])
9       )
```

Výpis 7: Chování metody drawBigScaleForSpecificPixel

Režim vykreslování v malém měřítku, tedy v situaci kdy uživatel vidí méně bodů, než kolik má k dispozici pixelů, funguje následovně. Z datové sady se určí data ke zobrazení. Polohy bodů měření v ose Y se opět berou z již vypočítané mapy `YPoints`. Polohy bodů měření na displeji v ose X se musí teprve určit. K těmto hodnotám se vypočte mapa se souřadnicemi. Vytvoření této mapy je popsáno v následujícím kódu.

```
1  xMapa = Map()
2
3  pocetDostupnychPixelu = (sirka-posunOsyY)
4  pocetViditelnychBodu = howMuchPointsIsVisible()
5
6  rozestupMeziBody = pocetDostupnychPixelu/pocetViditelnychBodu
7
8  volnaPozice = XSouradniceOsyY
9
10 for (x in 1 until pocetViditelnychBodu step 1) {
11     xMapa[x] = volnaPozice
12     volnaPozice += rozestupMeziBody
13 }
14 return xMapa
```

Výpis 8: Výpočet mapy bodů pro snímek v malém měřítku

Algoritmus vykreslování v malém měřítku potom spojuje jednotlivé body dle jejich souřadnic v ose X a ose Y.

Celý systém dvojího vykreslování má za následek optimální výkon při prezentaci dat uživateli. Nelze je navzájem zaměnit z následujících důvodů. Vykreslování ve velkém měřítku totiž spoléhá na zakreslení čáry z maxima do minima v daném úseku. V případě velkého množství dat je to velmi rychlý způsob vykreslení bez ztráty detailů, nicméně pokud nejsou k dispozici dvě nebo více hodnot pro každý pixel, nelze tímto způsobem realizovat přiblížení.

Druhý způsob kreslení, tedy kreslení v malém měřítku naopak spojuje jednotlivé body, což se hodí pouze pro přesné zakreslení dat v případě, že je třeba zakreslit méně bodů, než kolik je pixelů. Takto lze realizovat přiblížení. Použití tohoto kreslení ve formě spojování bodů v případě, kdy máme více bodů měření, než kolik je k dispozici pixelů, vede k problémům s výkonem, protože se potom zbytečně kreslí na ta samá místa.

Přechod mezi oběma variantami kreslení není násilný a graf si drží svou podobu v obou variantách. Srovnání k dispozici na obrázku 10 a obrázku 11.

Obrázek 10: Kreslení ve velkém měřítku

Obrázek 11: Kreslení v malém měřítku

3.4 Navržené aktivity

Aplikace implementuje dvě aktivity. Jedná se o `MainSelectionActivity` a `GraphActivity`. Popis aktivit z uživatelského hlediska, tj. popis funkcionality, je popsán v kapitole 5. Tato kapitola se bude věnovat stručnému technickému popisu aktivit. Obě zmíněné aktivity vychází z jednoduché koncepce MVC, viz. 2.5.1.

3.4.1 MainSelectionActivity

Cílem aktivity je výběr měření k následnému zobrazení. Na základě uživatelských vstupů v podobě výběru stanice a poté měření, průběžně stahuje dostupná data prostřednictvím fasády `DataHandler` viz. 3.2.1.

Pro výběr měření musí mít uživatel k dispozici zobrazená měření pro vybranou měřící stanici. Webová služba nevrací metadata, tedy seznam měření, na základě konkrétního data a konkrétní stanice, nýbrž pouze na základě časového razítka viz. tabulka 9, metoda `getMetadataType`. Dostupná metadata tedy filtruje podle stanic tak, aby uživatel viděl pouze seznam měření, které se týkají vybrané stanice.

Ke stahování sady metadat, tedy seznamu měření, aktivita používá implementovaný `DateSpinnerDownloadStack`, který rozhoduje o tom, zda je na místě stahovat novou sadu metadat. Uživatel totiž nechce stahovat metadata hned po zaznamenání události změny data, protože posouváním otočných částí `DateSpiner`, například po dnech, může dojít k vyvolání události stahování několikrát po sobě pro různé dny. Tomuto stavu předchází `DateSpinnerDownloadStack`, který je v podstatě implementací jednoduchého zásobníku, který si ukládá časová razítka změn dat a dovolí stáhnout nová metadata, až když od poslední změny uběhne 1500 milisekund.

Po identifikaci konkrétního měření ze seznamu se spouští stahování dat ke zobrazení. Všechna potřebná data, která mají být přenesena do další aktivity, jsou zabalena do datového objektu třídy `StationInfoContainer`, který je předán objektu třídy `Intent`. Prostřednictvím objektu třídy `Intent` je poté spuštěna další aktivita `GraphActivity`. Jedná se o standardní postup přechodu z jedné aktivity do druhé[10].

3.4.2 GraphActivity

Jde o aktivitu, která obsahuje pouze komponentu `GraphComponent` a jednoduché menu aktivity. Menu koresponduje s již popsanými možnosti mimo uživatelská gesta na konci kapitoly 3.3.1. Hlavní cíl aktivity je pouhé zobrazení komponenty a zpracovávání vstupů a případná prezentace detailů o stanici a měření, které nejsou nijak spojeny s komponentu samotnou. Pokud tedy pomineme komponentu `GraphComponent` jako takovou, aktivita umí pouze zobrazit dostupné informace o měření formou dialogu, který reflektuje hodnoty typu `string` pro zobrazení. Jako další implementovaná externí funkcionalita je zobrazení a editace anotací, vše o anotacích je v kapitolách 3.1.4 a 3.1.5.

Pro zobrazení komponenty `GraphComponent` je nutné ji nejprve inicializovat. Ukázka inicializace je již k nahlédnutí v rámci kapitoly 3.3.3 a to včetně rozebrání konstruktoru. Konstruktor totiž potřebuje předložit nějaká data. Ve zmíněné kapitole 3.3.3 je dostupná jak ukázka kódu, tak seznam parametrů v tabulce 11. Zvýrazněné chybové body mohou být předloženy konstruktoru v podobě seznamu, viz. seznam `errorPoints`. Webová služba s hotovým klasifikátorem pro

[10] https://developer.android.com/reference/android/content/Intent

získání těchto chybových bodů je stále ve vývoji a proto dochází k pouhé simulaci identifikace těchto bodů.

Simulaci má na starosti aktivita `GraphActivity`. Tato aktivita simuluje identifikaci chybových bodů, které jsou zobrazovány jednoduchým a nepřesným způsobem. Jedná se pouze o dočasnou simulaci, která nebere v potaz reálná data a slouží pouze jako demonstrace toho, že je komponenta `GraphComponent` na zobrazení chybových bodů připravena.

Cílem práce nebyla identifikace chybových bodů jako taková, a proto nebylo přistoupeno k náročnějším výpočtům. Dochází tak do určité míry zároveň k prevenci působení na výkon aplikace externími vlivy. Bude potom snadnější ohodnotit, jak je efektivní aplikace s implementovanou komponentou. Jednoduchý algoritmus si rozdělí datovou sadu na bloky o velikosti 20 000 na jeden blok a dané bloky projde. V případě, že se v bloku nachází měření na jedné ze 4 fází, které má vyšší hodnotu než 50, dojde k zaznačení chybového bodu. Výsledný seznam chybových bodů je předán při inicializaci komponenty.

4 Testování aplikace

Aplikace byla testována z několika pohledů. Bylo testováno, zda aplikace splňuje funkční i nefunkční požadavky, které byly konzultovány na začátku a v průběhu projektu. Po dokončení vývoje aplikace a provedených akceptačních testech lze prohlásit, že aplikace splňuje všechny akceptační kritéria a to včetně dílčích požadavků na komponentu.

4.1 Využití hardware

Výsledky měření využití hardware jsou k dispozici na obrázku 12 jedná se o minutový test, který spočíval ve spuštění aplikace, vybrání měření a maximálním přiblížení s posuny. Test proběhl na zařízení Xiami Mi A2. Toto zařízení je osazeno displejem o rozlišení 2160 x 1080 pixelů, využívá 4096 MB RAM, procesor Qualcomm Snapdragon 660, který je vybaven osmi jádry o maximálním taktu 2.2 GHz.

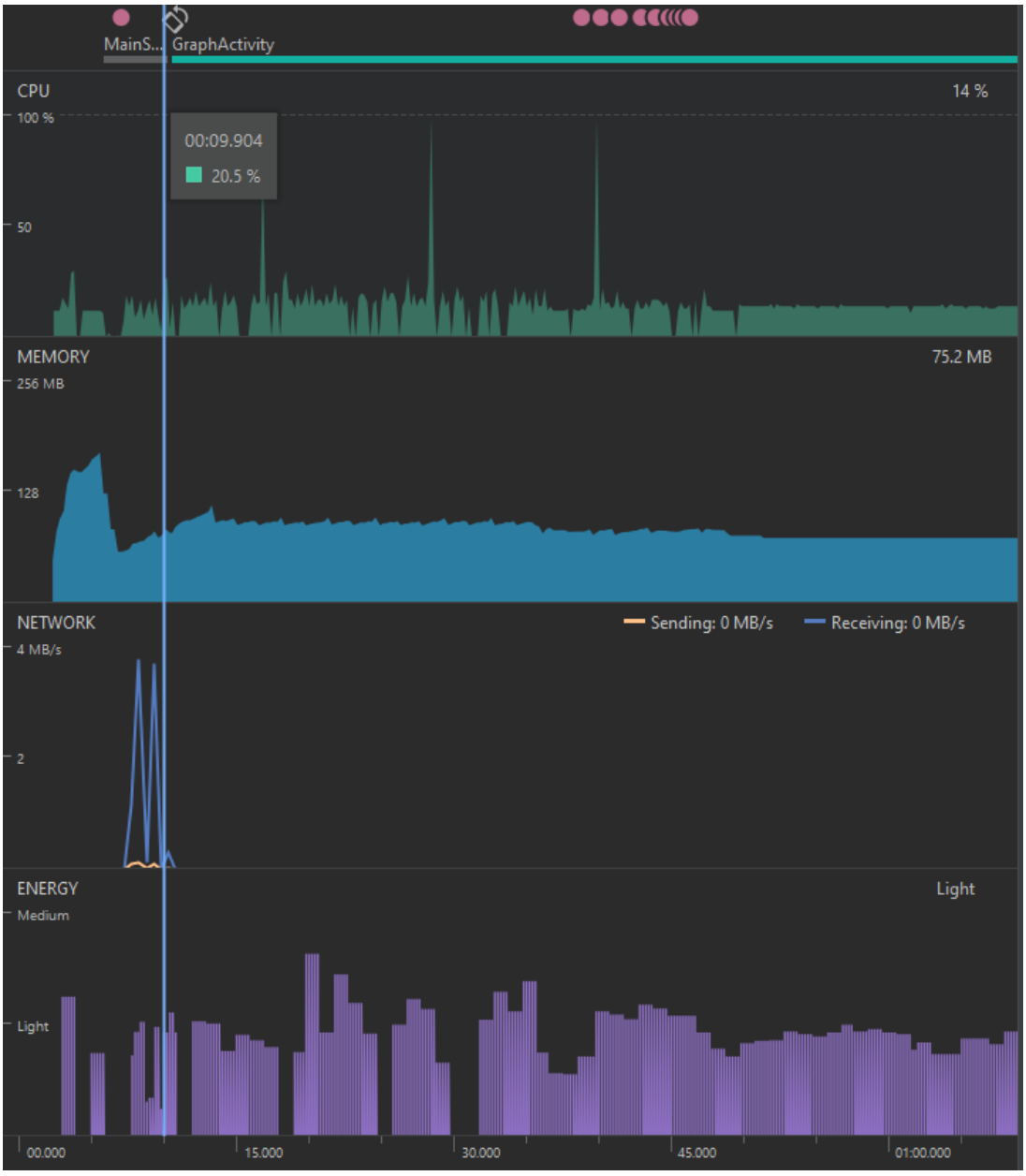
Dle výsledků měření se využití procesoru pohybovalo okolo 20% při aktivním využívání aplikace a okolo 14% mimo aktivitu uživatele. Aplikace tedy nevyužívá procesor nijak zásadně. Větší část využití procesoru ve chvíli testu pocházela z ostatních aktivit mobilního telefonu. Jedná se o zelený graf v horní části obrázku 12. Na obrázku lze pozorovat tři vrcholy, které se projevily pouze v řádech milisekund a jejich původ se pravděpodobně nachází mimo testovanou aplikaci, neboť se nepodařilo stejného výsledku docílit opakovaně.

V další části obrázku 12 lze pozorovat modrý graf využití operační paměti. Aplikace drží využití operační paměti v rozmezí od 75 do 80 MB.

V rámci síťového provozu lze rozpoznat stahování dat, jedná se o sekci označenou jako „NETWORK“. Objem stažených dat pro kompletní otevření jednoho měření je 4.68 MB. Ani zde se nedá mluvit o nadbytečném přístupu k síti. Aplikace si stahuje pouze to, co k otevření jednoho měření potřebuje.

Sloupcový fialový graf na obrázku 12 symbolizuje analyzátor využití energie. Aplikace se drží většinu času v označení lehkého využité baterie což koresponduje s povahou a účelem aplikace.

V případě, že uživatel není aktivní, aplikace pouze čeká na další vstupy. Neprobíhá vykreslování ani žádné výpočty.



Obrázek 12: Měření využití hardware

4.2 Výkonnostní srovnání

V tabulce 11 lze vidět demonstraci toho, o kolik je nová komponenta lepší, než využití komponenty třetí strany, viz. 3.3.

Tabulka 11: Srovnání aplikací

Typ testu	**Balíček com.jjoe64:graphview**	**Nová komponenta**
Otevření aplikace při mobilním 4G připojení - 5 Mbps	12 vteřin	5 vteřin.
Otevření aplikace při WiFi - 25.34 Mbps	9 vteřin	3 vteřiny
Plynulost vykreslování	Od 0.5 FPS do 15 FPS v závislosti na měřítku.	Plynulých 25-30 FPS.

V rámci měření FPS muselo dojít ke kompromisu u použití knihovny třetí strany. Nebyla možnost použití žádné formy měření FPS přímo z komponenty jako takové, u tohoto měření se tedy jedná o odhad. Ve velkém měřítku docházelo k zásekům i na dvě vteřiny. Aplikace nepřekonala 1 FPS až do měřítka, které odpovídalo 10 000 zobrazených bodů. Mírné zlepšení nastupovalo při dalším zmenšování měřítka a to až do chvíle, kdy bylo viditelně méně bodů, než kolik bylo k dispozici pixelů displeje. Tedy rozmezí zhruba 1 000 viditelných bodů. Při dalším zmenšování se hodnota blížila odhadnutým 15 snímům za vteřinu.

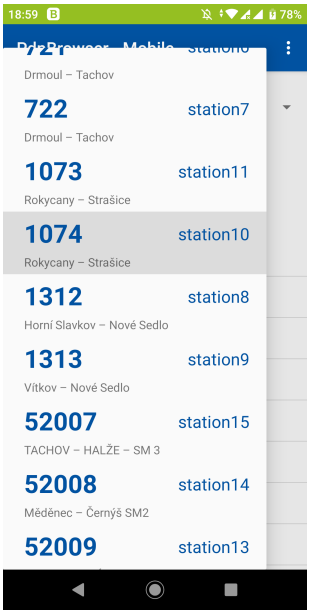
5 Uživatelská dokumentace

Následující kapitoly budou věnovány uživatelské dokumentaci. V této části se uživatel dočte, jaké má aplikace možnosti, co dokáže a jak ji má obsluhovat.

5.1 Výběr měřeni

Obrázek 13: Výchozí stav aplikace

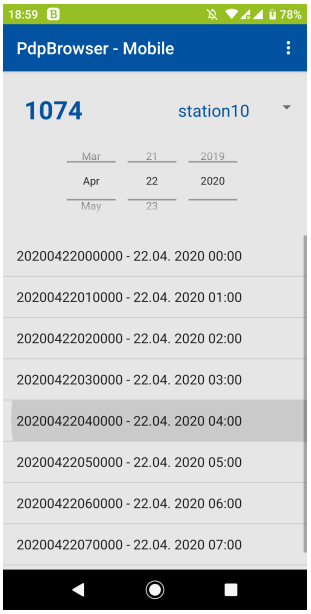
Po otevření aplikace vypadá jako na obrázku 13. Nahoře je k dispozici možnost vybrat stanici. Na obrázku 13 je již vybrána stanice 1074 station10. Po kliknutí na toto pole se zobrazí seznam stanic stejně, jako tomu je na obrázku 14. Každá stanice má uvedeno její identifikační číslo, název a svou lokaci.



Obrázek 14: Výběr stanice

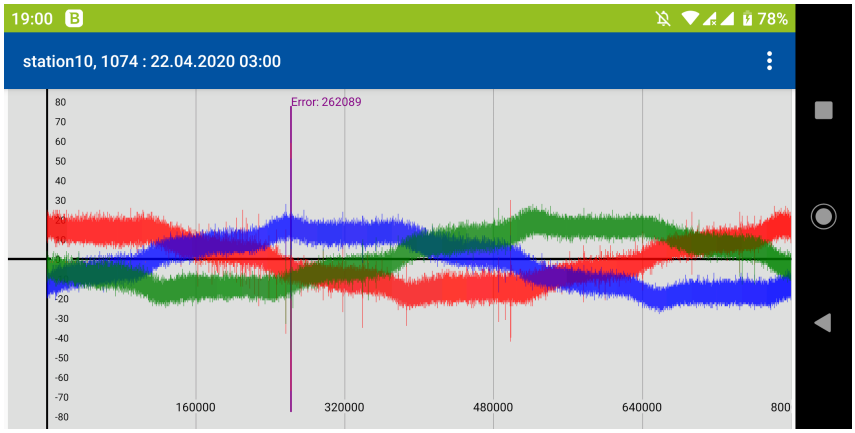
Prostřednictvím tří teček v pravém horním rohu aplikace je k dispozici otevíratelné menu. Toto menu obsahuje pouze jednu položku a tou jsou informace. Po otevření dojde k zobrazení základních informací o verzi aplikace a jejím autorovi.

Pro otevření měření je třeba jej vybrat s hlavní nabídky, stejně jako tomu je na obrázku 15. Po kliknutí na jakékoliv měření se zahájí stahování. Ve chvíli, kdy jsou potřebná data stažena, dojde k otevření měření.



Obrázek 15: Výběr stanice

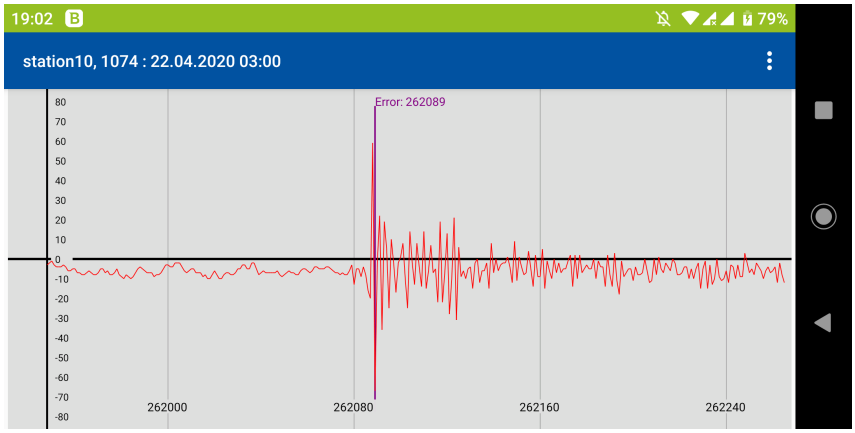
5.2 Otevřené měření



Obrázek 16: Otevřené měření

Po otevření měření aplikace vypadá jako na obrázku 16. Většinu obrazovky zabírá vykreslené měření. Pro přehlednost je uvedeno datum a čas měření, včetně stanice, ze které měření pochází. Ve výchozím stavu jsou viditelné všechny čtyři fáze. Měření je zakresleno do popsané osy X a osy Y. Fialovou barvou je zvýrazněný detekovaný bod poruchy.

Přiblížené měření v bodě poruchy je k nahlédnutí na obrázku 17. Pro vyšší přehlednost je vypnuto vykreslení ostatních fází.



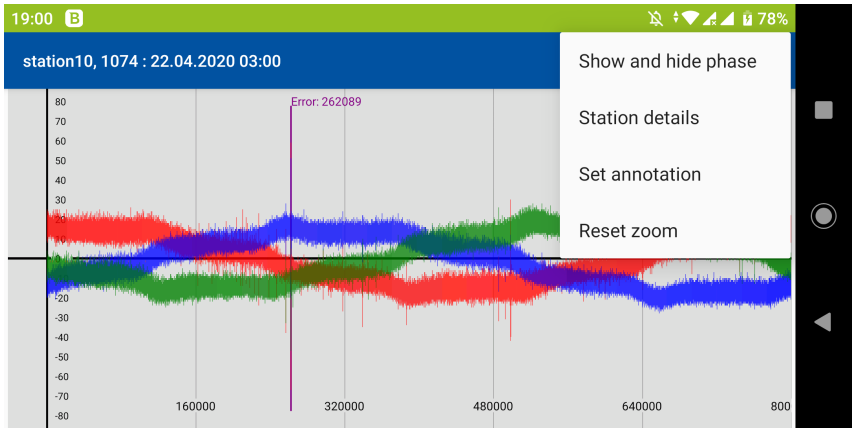
Obrázek 17: Otevřené měření

5.3 Ovládání grafu

S grafem lze manipulovat podobně jako při prohlížení fotografie. Pro přiblížení v určitém bodě stačí, aby uživatel provedl standardní gesto pro přiblížení, nebo oddálení, pokud se chce naopak od určitého místa oddálit. Pro pohyb směrem doprava nebo doleva stačí použít gesto pro pohyb, stejně jako tomu je při prohlížené přiblížené fotografie. Stačí tedy uchopit graf a táhnout jedním z požadovaných směrů.

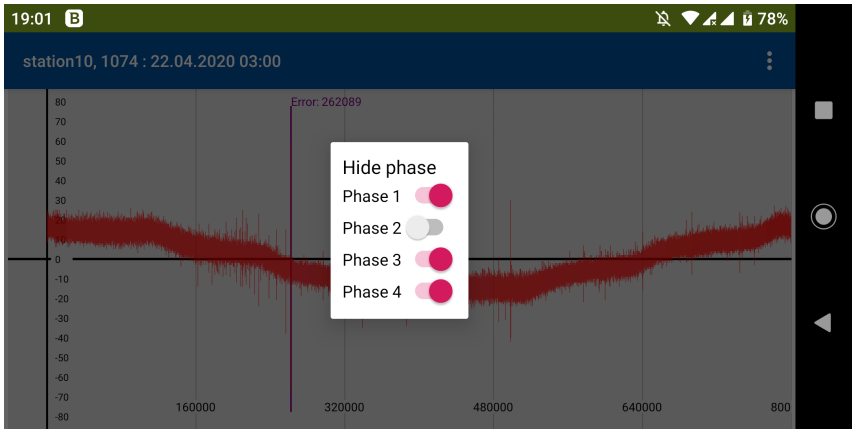
5.4 Uživatelská nabídka

Prostřednictvím tří teček v pravém horním rohu aplikace se uživatel dostane do nabídky, která dává možnosti, jenž jsou k nahlédnutí na obrázku 18.



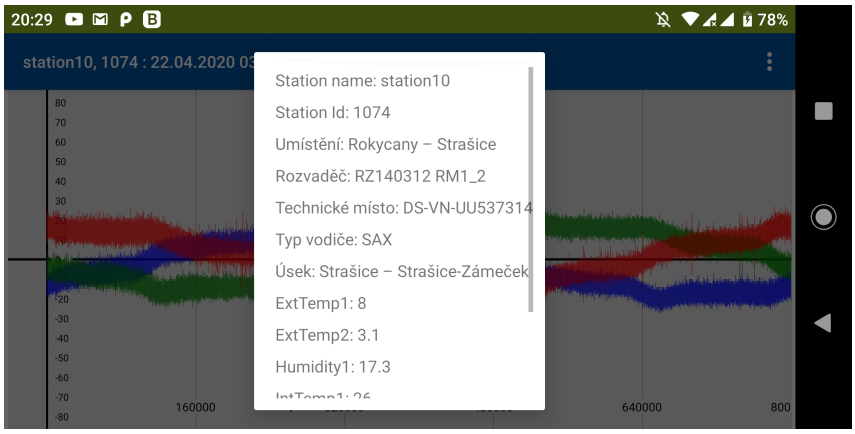
Obrázek 18: Otevřená uživatelská nabídka

Show and hide phase je možnost, která otevře konfigurační okno pro nastavení viditelnosti fází, viz. obrázek 19. V tomto konfiguračním okně jsou přepínače, které lze libovolně nastavit. Okamžitá odezva je viditelná skrze poloprůhledné okolí, tak jako tomu je na obrázku



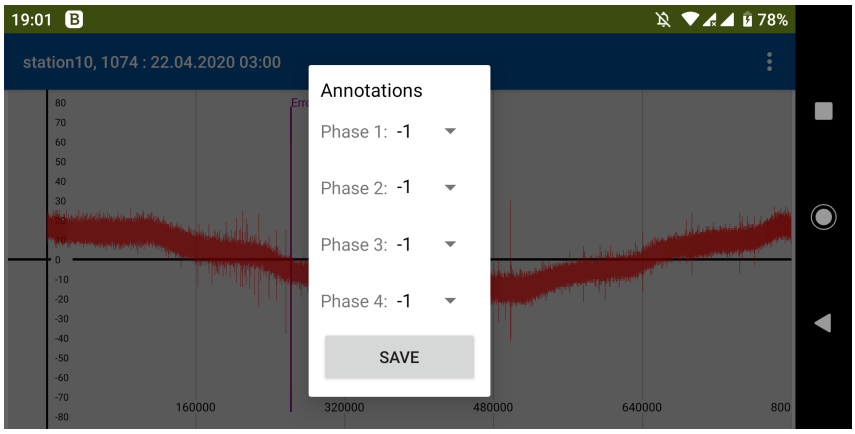
Obrázek 19: Otevřené konfigurační okno

Další možností v menu je Station details. Vybrání této možnosti zobrazí dostupné detaily o měření a stanici tak jako tomu je na obrázku 20.



Obrázek 20: Otevřené okno s detaily

Po vybrání předposlední možnosti, tedy Set annotaiton se zobrazí menu, které umožní uživateli měnit anotace pro jednotlivé fáze. Přesně jako tomu je na obrázku 21



Obrázek 21: Otevřené okno s anotacemi

Poslední možnost, kterou uživatel může zvolit, je Reset zoom. Tato možnost resetuje graf do původní podoby, tedy do podoby po otevření měření.

6 Výroba instalačního balíčku

Instalační balíček v podobě APK souboru lze snadno vytvořit prostřednictvím vývojářského prostředí Android Studio. Tento APK soubor slouží k instalaci aplikace. Manuální distribuce tohoto APK souboru je jedna z možností rozšíření aplikace technikům, nicméně není optimální. Mnohem vhodnější by byl přístup využití standardní distirbuční služby Google Play. Obrovským přínosem tohoto způsobu distribuce by bylo snadné řízení aktualizací a sběr případných chybových logů. Aplikace by šla šířit buď veřejně, pokud nebudou data v ní obsažená identifikována jako citlivá, nebo v režimu soukromé aplikace. Google totiž nabízí možnost publikování soukromých aplikací tak, aby byly dostupné pouze uživatelům v rámci konkrétní organizace.

Jakákoliv možnost distribuce aplikace prostřednictvím Google Play si vyžaduje vývojářský účet. Ten lze snadno založit za poplatek 25 Amerických dolarů, což je v době vzniku této práce asi 633 Českých korun.

7 Závěr

Přínosem práce jako takové má být vznik jak aplikace samotné, tak vznik komponenty pro zobrazení velkého množství dat bez vzniku jakýchkoliv ztrát. Tyto cíle byly naplněny. Výstupem práce byla funkční aplikace, která nabízí technikovi možnost plynulého průchodu grafu libovolného měření. Aplikace jako taková splnila všechny požadavky, které na ni byly kladeny. Do budoucna bude dále vyvíjena, neboť je stále prostor pro vylepšení, například v podpoře režimu bez připojení k internetu. V tomto zamýšleném režimu nebude třeba, aby technik měl k dispozici mobilní internetové připojení. To dá možnost prohlédnout již stažené měření i v případě práce v terénu, kde jsou problémy s pokrytím mobilním signálem.

Komponenta samotná také splnila všechny požadavky, které na ni byly kladeny. U zobrazených měření nedochází ke ztrátě dat a pohyby v něm jsou naprosto plynulé. Po uživatelském vstupu dochází k okamžité odezvě. Komponenta dává možnost plně intuitivního procházení grafu. Komponenta pro plynulý chod nevyžaduje zařízení s vysokým výkonem. Při průchodech grafu, a to zejména při přibližování nebo oddalování nedochází ke skákání či zasekávání. Graf se při přibližování pomalu roztahuje či smršťuje při přibližování.

Nevýhoda komponenty spočívá v malé generičnosti. Komponenta splňuje zadání práce a poradí si jistě s jakoukoliv délkou dat libovolného počtu datových sad, nicméně data v současné implementaci musí být v podobě seznamu typu byte hodnot. Plánované refaktorování s cílem podpory generičnosti bude snadné. Jakmile komponenta bude generická, bude vhodné publikovat ji jako knihovnu k volnému použití. Tím se přínos této diplomové práce ještě více rozšíří.

Literatura

1. KRÁTKÝ, Michal; MIŠÁK, Stanislav; GAJDOŠ, Petr; LUKÁŠ, Petr; BAČA, Radim; CHOVANEC, Peter. IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS. 2018, roč. 65, č. 1, s. 543–552.
2. DRAKE, Joshua J. *Android Hacker's Handbook*. 2014.
3. *Mobile Operating System Market Share Worldwide* [online] [cit. 2020-02-26]. Dostupné z: `https://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-200901-202001`.
4. *Android hacking pro začátečníky – architektura* [online] [cit. 2018-02-25]. Dostupné z: `https://www.hackingkurzy.cz/blog/android-hacking-pro-zacatecnik-arhitektura/`.
5. BORNSTEIN, Dan. *Google I/O 2008 - Dalvik Virtual Machine Internals* [online] [cit. 2020-04-04]. Dostupné z: `https://www.youtube.com/watch?v=ptjedOZEXPM&feature=emb_title`.
6. *tack based VM vs. Register based VM* [online] [cit. 2020-05-05]. Dostupné z: `http://beyondthegeek.com/2016/08/14/stack-based-vm-vs-register-based-vm/`.
7. *Understand the Activity Lifecycle - diagram* [online] [cit. 2019-12-27]. Dostupné z: `https://developer.android.com/guide/components/images/activity_lifecycle.png`.
8. *Kotlin on Android. Now official* [online] [cit. 2017-03-17]. Dostupné z: `https://blog.jetbrains.com/kotlin/2017/05/kotlin-on-android-now-official/`.
9. *Base standard* [online] [cit. 2020-05-05]. Dostupné z: `https://tools.ietf.org/html/rfc4648`.
10. *Object Expressions and Declarations: Object declarations.* Dostupné také z: `https://kotlinlang.org/docs/reference/object-declarations.html`.